コンピュータ・サイエンス研究書シリーズ ⑤

オペレーティング・システムへの構造的アプローチ（下巻）

日本アイ・ビー・エム㈱
江村潤朗 著

日本コンピュータ協会

コンピュータ・サイエンス研究書シリーズ

オペレーティング・システムへの構造的アプローチ（下巻）

日本アイ・ビー・エム(株)　江村潤朗 著

日本コンピュータ協会

まえがき

数年がかりで，やっとオペレーティング・システムの全構造をまとめあげることができました．最初は，雑誌「**ソフトウェア技術**」（日本コンピュータ協会・発行）の第2巻第2号以降に十数回にわたって連載しました．この時点では，まだオペレーティング・システムがさほど一般化しておりませんでした．したがって，ごく基礎的な事柄から論じてみました．この部分は，上巻の前半の部分の内容となりました．その他の部分は，折をみてまとめあげてゆきました．途中再三にわたって執筆を中断せざるを得ない状況に追い込まれましたが，曲りなりにも，一応全体をまとめあげることができました．この間，オペレーティング・システムは，その基本構造は堅持しながらも，幾多の新しい機能を加味しながら，どんどん成長してきました．今では，一人の人間でオペレーティング・システムの全貌を正しく把握し，詳細な個々の機能構造を理解するのは相当に困難なほど巨大化し，複雑化した感じがします．

2年間ほどアメリカに渡って，多重仮想記憶システム（multiple-virtual system）を基本としたオペレーティング・システムの研究に専念してきた私の同僚は，帰国してきてそのオペレーティング・システムに“怪物(モンスター)”という異名をつけました．それほど，複雑怪奇なものにみえるまで成長してしまったのです．しかし，オペレーティング・システムを学ぶための王道に近いものが存在すると思います．それは，オペレーティング・システムの基本構造をシステムとして理解することです．しかも，ニーズを原点に諸々の機能や相互の関連を追求してみることです．このことを念頭においてまとめあげたのが，本書（上巻，中巻，下巻）でもあります．オペレーティング・システムを本格的に研究される際の第一歩として十分役立つように配慮したのが本書です．

上巻では，オペレーティング・システムのシステムとしての基本構造を解明

し，それをささえるためのハードウェア構造に論及しました．さらに，オペレーティング・システムのもとでのプログラム設計構造と連係編集プログラムの機能構造，およびデータ管理の諸機能とその構造を検討しました．

中巻では，データ管理機能の支柱ともいうべき「データ・セットの編成法とアクセス方法」を具体的に解説しました．また，「ジョブ管理の諸機能とその構造」についても詳述しました．

本巻（**下巻**）では，３つのテーマをとりあげています．タスク管理，仮想記憶システム，複合システムの３つです．

システム資源の管理を主要なねらいとしたタスク管理に関しては，第８章「**タスク管理の諸機能と構造**」で，その基本構造をふまえたうえで，５大機能（割込み処理機能，タスク監視機能，主記憶装置監視機能，内容監視機能，タイマー監視機能）を主体に詳しく論じています．単に機能を具体的に紹介するだけにとどまらず，その内部構造にかなり立ち入って，各種制御テーブルだとかロジックについても説明しています．あえて内部構造まで立ち入ったのは，そうしないとタスク管理の機能が本質的に理解しきれないからであり，また，タスク管理の詳細を論じた書物がほとんど見当らないのを補うといったねらいがあったからです．極力具体性をもたせることに留意し，この章だけで100種類をこえる図版を挿入しています．

第９章「**仮想記憶システムの概念と構造**」では，新しくオペレーティング・システムに仲間入りした仮想記憶システムを正面からとりあげました．仮想記憶システムの出現の背景を明らかにしたうえで，一般的仮想記憶システムの概念的構造を検討したうえで，それを実現するためのハードウェア機能とソフトウェア機能について具体的に提示しています．ソフトウェアに関してはページ監視プログラムのロジック（アルゴリズム）について，かなり詳細にふれています．

最新のジョブ・スケジューラの一種ともいえる JES (Job Entry Subsystem) を中心に展開したのが，第10章「**ゆるやかに結合された複合システムの機能構造**」です．柔軟結合の複合システム (loosely coupled multi-processing system) ともいわれますが，“loosely”という原語にふさわしいように“ゆるやかに結合

された"という冗長な表現をあえてしました．ここでは，複合システムの出現の背景をおさえたうえで，複合システムの構成(ハードウェアとソフトウェア)と JES 3 の処理機能を主体に展開しています．今後のジョブ管理プログラムの一つの形態を示唆するものとして，JES 3 をかなり詳しく説明しています．なお，本章は，先輩であり同僚でもある佐々木乾一氏のアメリカでの研究成果にもとづいています．

これまで，本書の執筆にさいして終始激励しご協力くださり，発刊にこぎつけられました**日本コンピュータ協会**の村越勝弘編集長はじめスタッフの方々に感謝の意を表したいと思います．

最後に，本書の執筆中に全面的に協力してくれた3人の子供達（愛美，望，保寿）に心からお礼をいいたい．母を亡くしたショックがあるにもかかわらず，炊事，洗濯などの家事に全面的に協力してくれたおかげで，なんとか脱稿にこぎつけることができたといえる．想えば，上巻の執筆に着手したとき，長女の愛美は小学一年になったばかりだった．今はすでに小学校卒業を目前にひかえている．この間終始協力し援助してくれた愛妻芳子はもはやこの世にいない．今は亡き妻の心労と配慮に感謝し，心をこめて本書を霊前に捧げたい．

昭和51年3月

江 村 潤 朗

目　　次

第 8 章

タスク管理の諸機能とその構造

一. システム資源の効果的利用と管理

コンピュータ利用の初期の段階の最大の関心事は，解決すべき問題を面倒な機械語で正しくプログラム化することにあった．とにかく，コンピュータが目的とした作動をし，正確な結果を出してくれれば一応満足であった．手作業で問題解決を行なうよりもはるかに早いのであるから，それなりの満足が得られたのである．しかし，除々にコンピュータの利用範囲が拡大され，解決すべき問題が複雑化し，コンピュータにかかわりを持つ人間が増えてくると，機械語でプログラムを書くことのわずらわしさが意識されるようになってきた．そこで，プログラミングの負担を軽減するための各種の努力が払われた．その結果，記号言語や高水準言語が開発され，プログラミング・システムとして結実していった．また，さまざまな汎用プログラムの開発も行なわれた．

人間の欲望は留まることを知らない．コンピュータ利用も，プログラミング言語の出現によって，ますます広範囲の分野への活用が促されることになった．高度で複雑な分野へとその適用がおし進められていったことはいうまでもない．一方，ソフトウェアの技術的進歩の背景には，ハードウェアの長足の進歩があったことも見逃せない．

このような急速なコンピュータの発展の結果，多くのコンピュータ部門が直面した問題は，大規模な利用分野の拡がり，複雑さ，高度化が増したばかりでなく，定型的な処理業務の数も非常に増加したことである．こうなってくる

と，必然的にコンピュータ・システムに無駄が生じていることが気になりはじめてくる．ハードウェア資源，情報資源あるいは人的資源の生かし方が問題となる．この問題に対処するために，システム資源の効果的利用の工夫が行われるようになってきたのである．

1.　システム資源の種類

コンピュータ部門の諸資源は多大の投資に相当する．したがって，ハードウェア資源（ないしはソフトウェア資源）および人的資源など，コンピュータとかかわりを持つシステム諸資源を効果的に活用しなければならない．そのためには，各々の資源をそれぞれに最もふさわしい仕事に常時ふり向けておく必要がある．CPUが遊休することなくデータ処理という単調な仕事に専念できるためには，ハードウェア資源と情報資源が何時でも利用できる状況を作り出しておかなければならない．人的資源を効果的に活用するためには，コンピュータが代行しうる仕事から解放して，もっと人間特有の創造的な仕事，つまりコンピュータの新しい利用法を考え出したり，問題解決の公式を作りあげたり，条件の変化や予期しない事態に対処したり，コンピュータ室の管理などの本来的仕事に従事できる体制を確立しなければならない．ともあれ，まずはこれらシステム資源の内容を明らかにしてゆこう．

コンピュータでのデータ処理を効率よく行なうには，ハードウェアの構成要素を必要なときに直ちに利用できる状態にしておかなければならない．この場合のハードウェア構成要素，つまりハードウェア資源とは次のようなものをいう：

- 中央演算処理装置の使用時間（CPUタイム）
- 使用可能な主記憶装置の容量
- 入出力装置
- 入出力チャネルの使用時間
- 使用可能な補助記憶装置の容量

これらハードウェア資源の中で最も重要なものが，CPUで処理を行なうため

に使用することのできる時間である．CPUで行なう処理とは，いうまでもなくデータを処理することである．四則演算，主記憶装置のある場所から別の場所へのデータの移動，ある形式から別の形式へのデータ変換あるいは判断操作のための論理演算などを行なうための時間が**CPU 時間**である．

連続した長時間の処理を人手を借りずに，かつ CPU の遊びを出来るだけ少なくして行なうためには，CPUからすぐにアクセスできる形式でプログラムとデータを貯えておくための資源が必要である．このために主記憶装置があり，そこから迅速に情報を入手したり，処理の結果をそこに戻したりすることができるようになっているのである．

また，当然のことながら，この主記憶装置にプログラムやデータを迅速に入れる方法がないと困る．さらに，主記憶装置の容量には限りがあるため，CPUで処理した結果をはき出しておくための装置が必要となる．したがって，データを連続的にかつ自動的に処理してゆくためには２つの資源がさらに必要である．つまり，情報を読み取ったり記憶したりするための**複数個の入出力装置**と，主記憶装置との間で情報を迅速に転送するための**入出力チャネル**とが必要である．

情報をチャネルを通して主記憶装置へ送り込んだり，主記憶装置から取り出した情報をチャネルを通して書き出したりするために用いられる装置が**入出力装置**と呼ばれることは，いまさらいうまでもないことである．しかし，これら入出力装置は，その用途から２種類に分れる．磁気ドラムまたは磁気ディスク記憶装置などの直接アクセス装置は，主として主記憶装置に収容しきれない情報を記憶するのに使われるので，通常，**補助記憶装置**と呼ぶ．一方，カード読取装置，印刷装置などは，主としてシステムに情報を入力したり，システムの出力を記録したりするのに使用されるので，**入出力装置**と呼ばれる．このように，入出力装置と補助記憶装置の記憶スペースとは，本来の役割がかなり異るので，この２つは切り離して別個の資源として取り扱うのが普通である．

コンピュータでは，ハードウェア資源を利用するために二種類の情報資源が必要である．コンピュータは，人間が作成した一連の処理手順，つまりプログラムにしたがって作動する．プログラムは命令の集まりから構成される．この

命令が1個ずつ CPU に取り出され，解読されて指示が出されることによって，ハードウェア構成要素が作動する仕組みになっている．したがって，プログラムを構成する一連の命令はコンピュータにとって欠くことのできない情報資源である．一方，命令の指示によってデータが処理される．CPU で処理されるデータも，その性質から資源といえる．これらプログラムとデータが情報資源であり，CPU で何時でも利用できる状態になっていないと効率的なコンピュータ利用の実現が期待できないのである．

前述のハードウェア資源および情報資源と同じように重要なのが，コンピュータ室の人的資源である．**人的資源**とは，コンピュータ部門のスタッフとしてシステムを使用する人々のことである．コンピュータ室によって職務の分担はそれぞれ異っているが，通常つぎのような仕事を行なうグループが存在し，コンピュータ室の責任者の指揮にもとづいて仕事を遂行している：

- コンピュータ室の生産性を高めることを目的として，ソフトウェア・システムの計画，生成，保守，拡張およびコントロールなどの仕事を行なう人
- コンピュータ・システムに適用する新しい仕事を設計し，これらの仕事をコンピュータで処理するために必要な手順とプログラムを開発する人
- コンピュータ・システムの操作を担当する人

コンピュータ室が成果をあげるかどうかは，まさにこれら人的資源をいかに効果的に活用するかにかかっているといってもよい．

コンピュータ室の生産性は，これまでに概観したハードウェア，情報，および人的資源のそれぞれがいかに適切に選択され，各々の仕事を円滑に行ないうるように配置し活用するかによってきまる．現在のコンピュータだと，1分間に数十億回ものオペレーションを人間よりもはるかに高い信頼度で行ないうる潜在的可能性をもっている．しかし，このような高性能を発揮する可能性をもったコンピュータが行なうすべての仕事を計画し，その仕事を遂行するにあたって起こり得る偶発的事象の対処の仕方をきめるのは人間である．要するにコンピュータは人間が作りあげたプログラムに忠実に従うだけである．プログラムで処理の仕方を克明に指示しないかぎり，コンピュータは何の役にも立たな

いのである．

しかし，プログラムがあれば，コンピュータは単に入力データを処理し，出力データを記録するといった仕事以上の高度で複雑なことを行なうことができる．プログラマが，コンピュータ・システムの限られた命令を自由自在に組み合せ，一定の処理手順を構成することによって，かつて人間しかできなかった多くの事柄をコンピュータに代行させることが可能となる．また，場合によっては人間の手作業では実現できなかったような問題解決さえ行なわせることができる．これらの中には，プログラムの言語翻訳，システム資源の管理，情報検索，ジョブのスケジューリングとその実行の監視，ハードウェア装置の操作とコントロールなどの特別な仕事も含まれている．

このような特別な仕事を行なうことをコンピュータに指示するプログラム群が，いうまでもなく**オペレーティング・システム**である．オペレーティング・システムは，コンピュータの高度なシステム適用業務の１つであり，プログラムとデータとを組織的に統合することによって，システム資源を効果的に活用し，その結果全体的な生産性を高めることをねらいとしている．一般のコンピュータ適用業務と同じように，オペレーティング・システムも複雑な仕事を処理するために工夫されたものであるが，その仕事の対象と内容が異っている．つまり，一般の適用業務はコンピュータ室以外で発生する業務，たとえば銀行業務，プロセス制御，ミサイル設計などの問題を対象としている．これに対して，オペレーティング・システムは，コンピュータ室で発生している問題，つまりコンピュータ室に存在する諸資源を効果的に活用する問題に挑戦しているのである．要するに，オペレーティング・システムは，一般の適用業務の処理効率を高め，それをコントロールすることを目的に特別に設計されたソフトウェアのシステムである．コンピュータ・システムの諸資源を効果的に生かすことによってシステム全体の性能を高め，人的資源にとっての使い易さを増大させることこそが，オペレーティング・システムの第一の目的なのである．

2. 資源管理を行なう監視プログラム

オペレーティング・システムは，基本的には2種類のプログラムに大別できる．その1つは，ハードウェア資源の1つである CPU が**監視プログラム状態**（supervisor state）にあるときに実行されるプログラムである．もう一方は，CPU が**問題プログラム状態**（problem program state）にあるときに実行されるプログラムである．これら監視プログラム状態と問題プログラム状態に関しては「第3章オペレーティング・システムのためのハードウェア構造」ですでに説明したことであるが，システム資源の管理という視点から再度まとめておくことにする．

監視プログラム状態で実行されるプログラムのことを一般に**監視プログラム**（supervisor）と呼ぶ．この監視プログラムはオペレーティング・システムが提供するサービスおよび各種資源の中枢的役割りを果す．その主要な機能は，問題プログラム状態で実行されているプログラムが要求する種々のサービス，たとえば主記憶装置のスペースの割り当て，入出力オペレーションの実行，主記憶装置へのプログラムのローディング，プログラムの実行の開始などを行うことである．監視プログラムは，これらの機能を果すために，問題プログラム状態のプログラムの実行中に生じる割込みによって，CPU の制御権を受けとる仕組みになっている．**割込み**（interruption）には，問題プログラム状態のプログラムが能動的に特定のサービスを要求することによって生じる場合と，コンピュータ・ハードウェアによって自動的に引き起される割込みの場合とがある．

特定サービスの要求は，問題プログラム状態のプログラムが**監視プログラム呼出し命令**（supervisor call 命令；略して **SVC 命令**）を出すことによって行なわれる．このSVC命令が実行されるとSVC割込みが生じ，サービスを要求したプログラムの実行は一時中断され，制御権が監視プログラムに移される．なお，SVC 命令では，監視プログラムが要求するサービスに応じるために必要となる情報をパラメータとして与えるのが普通である．たとえば，主記憶装置にプログラムをローディングさせる要求の場合だと，ロードすべきプログラ

ムの名前をパラメータとして監視プログラムに受け渡す必要がある.

一方，自動的な割込みは，問題プログラム状態のプログラムが特定のサービスの要求をすることによって引き起されるものではなく，コンピュータ・ハードウェアによって起されるものである．この自動的な割込みは，コンピュータの内部に重要なあるいは異常な事象が生じたとき，たとえば入出力オペレーションの完了であるとか，ハードウェアやソフトウェアの誤りの発見などによって引き起される．つまり，監視プログラムによる援助を必要とするような事象が生じたときに，処理中のプログラムの実行を一時中断し，CPU の制御権を監視プログラムに渡すことができるように，ハードウェア自身が絶えず自分のオペレーションを監視しているのである．このような割込みのからくりのおかげで，問題プログラム状態のプログラムや監視プログラムが，特別な事象が生じたかどうかを判定するために絶えず検査したり，事象が起るのを無駄に待っているといったことを回避することができる．その結果，ハードウェア資源や情報資源の遊びを最小限にするよう仕向けることが可能となる．

いずれの割込みが生じても，CPU は監視プログラム状態に切りかわり，CPU の制御権は監視プログラムに渡される．さらに，割込みが生じた原因と割込みが生じた時点の CPU やプログラムの状況を示す情報は**プログラム状況ワード**（Program Status Word；略して **PSW**）におさめられ，監視プログラムに伝達される．監視プログラムが制御権を受けとると，すぐに割込みの原因となった事象が何であるかを判定するための分析を行い，その上で適切な措置をとる．その措置を構じたあとで，監視プログラムは PSW を利用して割込みが生じたプログラムの中断点の次の命令に制御権を返すことができるのである．ところで，多量プログラミングの状況では，該当プログラムとはまったく関係のない事象が原因で割込みが起る場合もある．たとえば，同時に実行されている別のプログラムのための入出力オペレーションが完了したことによって，目下 CPU の制御権を得ているプログラムの実行が割込みによって中断される．この場合には，監視プログラムの所定の処理のあと，割込みによって中断したプログラムに制御権が戻されることもあるし，入出力オペレーションが完了した別のプログラムに返されることもある．

割込みが生じて制御権が監視プログラムに渡ると，監視プログラムはある種の特別な命令，たとえば入出力命令や記憶保護命令などを CPU に実行させることができる．これらの命令は，監視プログラム状態のときだけ，つまり監視プログラムが制御権をもっているときだけ実行できるものであるために，**特権命令**（privileged instruction）と呼ばれる．この特権命令が問題プログラム状態のプログラムの一部として実行されると，他のプログラムや監視プログラムそのものの実行を妨げる危険性がある．したがって，問題プログラム割込みが生じるように配慮されている．

このような特権命令と割込みの仕組みは，監視プログラムが問題プログラム状態のプログラムからのサービス要求に応え，コンピュータ・システムで行なう仕事の実行を動的にコントロールし続けることを可能にする．処理プログラムが実行されている間，監視プログラムが処理プログラムのサービス要求に応じたり，コンピュータ・ハードウェアによって発見された事象に対処したりするために，CPU の制御権は絶えず問題プログラム状態のプログラムと監視プログラムの間を往き来しているのである．

監視プログラムは，ハードウェアとソフトウェア・システムの諸資源の割り振りとその使用の仕方に関して完全にコントロールしているのである．これを達成するために，監視プログラムはシステムに存在するすべてのプログラム資源，データ資源およびハードウェア資源の正確な利用状況をたえず記録保存している．さらに，監視プログラム自身の変化する状況も記録している．これによって，システムの活動をたえず制御し，あるプログラムが他のプログラムを妨害することがないようにし，システム資源の効果的利用の面倒をみてくれているのである．あらゆる出来事の最新の記録を保持しているおかげで，監視プログラムが CPU の制御権を断続的に受けとっても，システム資源の監視を統一的に行なってゆくことができるのである．

ところで，監視プログラムの主要な部分は，つねに主記憶装置内に入っている．中でも，入出力オペレーションを開始させ，その制御を行なう入出力監視ルーチンなどは，システムのオペレーションを連続的に行なわせ，バランスのとれたコントロールを実現してゆくために，主記憶装置に常駐している．ま

た，主記憶装置のスペースの割り振りを面倒みる監視ルーチンなども，使用頻度が高いのでやはり主記憶装置内につねに入れておかれる．一方，あまり使用頻度が高くない監視ルーチンだとか，主記憶装置内に常駐していなくてもシステムの効率的なオペレーションに支障をきたさない監視ルーチンは，その特定機能を必要とするときだけ，補助記憶装置から主記憶装置に呼び込まれる仕組みになっている．

監視プログラムは，これまで概観してきたようにオペレーティング・システムの一部としてシステム資源の管理上中心的で，かつ不可欠の役割りを果す．しかも，監視プログラムが行なう機能の大半はタスク管理の機能である．一部は，データ管理の機能とみなされるが，これらは「第5章データ管理の諸機能とその構造」で詳述した通りである．

3. 監視プログラムの主要な機能と資源管理

タスク管理（task management）とは，もともと監視的な機能のことである．したがって，タスク管理の機能を果すルーチンは監視プログラム状態で働き，一般には**監視プログラム**という言葉で総称されている．つまり，監視プログラムがシステムに存在するすべてのタスクの制御を行なうのである．監視プログラムの主要な機能は，前述したように問題プログラムから要求されたさまざまなサービスに対処することである．つまり，システム資源の効率的かつ調和のとれた利用を計るために，問題プログラムから要求されたサービスの遂行を調整し，制御するのが主要な機能である．監視プログラムは，目下実行中のプログラムが他のプログラムや制御プログラムのオペレーションを妨害することがないように調整する．このことは，監視プログラム状態でしか実行することが許されていない特権命令を活用することによって達成されるのである．

ともあれ，上記のような機能を果たしてゆくために，監視プログラムは**図8-1**で示したような手順で実行される．割込みが生じることによって自動的に現 PSW が新 PSW に置きかわる．これによって，監視プログラムの**割込み処理ルーチン**（interruption handler）に制御権が移る．言うまでもなく，割込みはオペレーティング・システムの他の部分ないしは問題プログラムからの特別

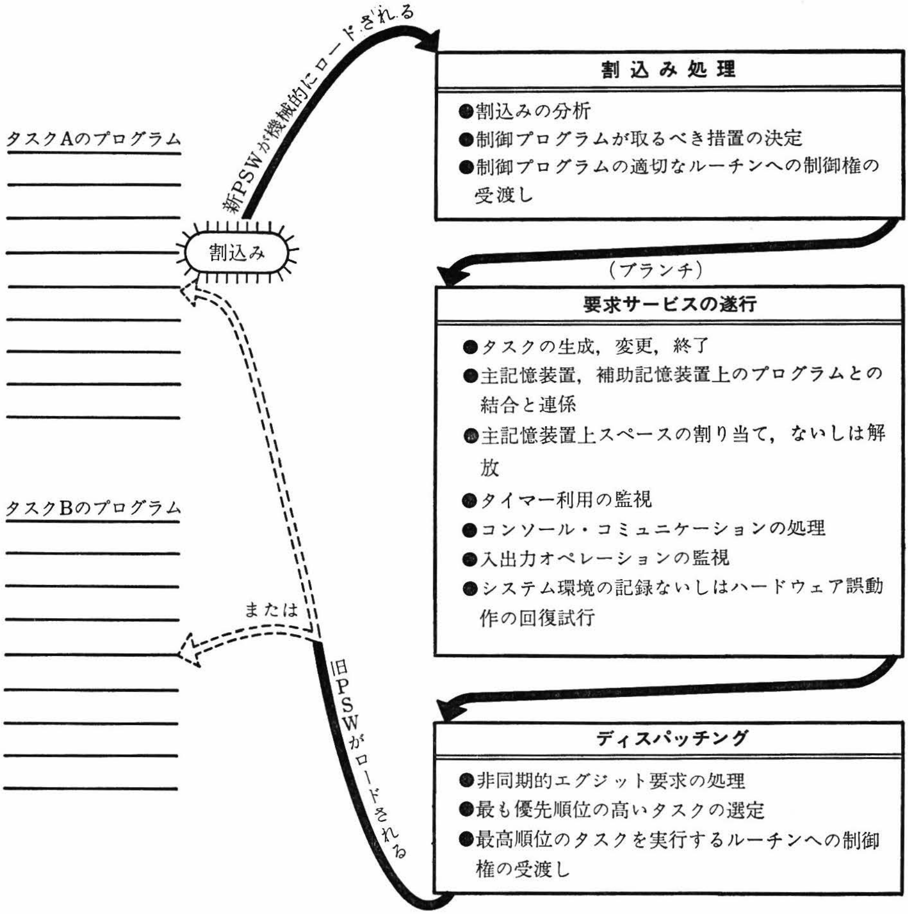


図 8-1　監視プログラムの全体的流れ

なサービス要求の結果引き起される場合と，入出力オペレーションの完了によって引き起される割込みのように自動的に生じる場合とがある．いずれの場合にも割込み処理ルーチンに制御権が移り，割込みの原因が分析され，それに見合った処理内容が決定される．そのうえで，割込み処理にふさわしい監視プログラム部分に制御権を受け渡す．その結果監視プログラムは，つぎに示すようなサービスの中から割込み原因にふさわしい処理を選んで実行することになる：

- 主記憶装置のスペースの割り振り
- 所要のプログラムないしはルーチンの主記憶装置へのローディング
- 主記憶装置へローディングしたプログラム ないしは ルーチンの使用スケジューリング
- あるタスクから別のタスクへの CPU 制御の切り換え
- あるタスクが別のタスクを妨害することの防止
- あらかじめ定められた優先順位に応じたタスクの同時処理と制御
- 同時に主記憶装置に存在する必要のない ルーチンに対する 主記憶域の共同利用の制御
- 時刻だとかタイミングに関するサービスの提供
- 入出力装置の誤作動に関する記録保持
- 入出力オペレーションのスケジューリングと監視

このようなサービス機能を遂行したあとで，制御権は監視プログラムのうちの**ディスパッチング・ルーチン**（dispatching routine）に渡る．このルーチンではタスクに関する後始末をしたうえで，最も優先順位の高いタスクを実行するルーチンへ制御権を戻すのである．

監視プログラムの機能のうち資源管理という観点からみて見逃すことができないのが，２つ以上のデータ処理用タスクを同時に実行し，監視してゆく機能であり，これはシステムの諸資源を効果的に管理することによってはじめて達成することが可能な機能である．以下で，システム資源の共同利用とその管理の仕方について概観しておくことにしよう．

プログラムは，タスクを実行するのに必要ないくつかの資源の中の１つにすぎない．データを入力し記憶したり，結果を出力し記憶したりするためには，入出力装置ないしは直接アクセス記憶装置のスペースが必要である．これもタスクを実行するための資源の１つとして位置づけられる．データを主記憶装置に出し入れしたり，入出力オペレーションを開始させコントロールするためには入出力チャネルの時間が必要である．これもシステム資源として不可欠だ．また，一連の命令（つまり，プログラム）およびこれらの命令が実行されるこ

とによって処理されるデータを記憶するための主記憶域も必要である．さらに，命令を実行し，それによってデータ処理を行うためには CPU タイムが必要である．これらもタスクを実行するうえで不可欠な資源である．

ところで過去のコンピュータの多くは，一度に1個のタスクしか実行しなかった．この場合には，実行中のタスクが完了するまで別の新しいタスクを始めることはない．したがって，プログラムの実行中のどの瞬間をとってみても，システムの使用可能な全資源の一部しか必要としない．その結果，システムの多くの部分がかなり遊んでしまう．たとえば，システムに備わっている主記憶装置および入出力装置の一部と，CPU のごくわずかの時間だけが使用されているに過ぎない．これでは，システム資源の有効利用は期待できないし，無駄があまりに多すぎる．しかも，このように一度に1個のプログラムしか実行しない環境では，タスクという概念はほとんど意識しなくてよい．1個のタスクで必要な資源を自由に使用できるので，プログラムが主記憶装置，CPU チャネルあるいは入出力装置やデータを使用するといったプログラム中心の考えで十分通用するし，何ら支障もなかった，監視プログラムがタスク単位に資源の割り振りをし，制御してゆく必要性はないのである．

一方，現在のコンピュータでは，複数個のタスクを同時に実行し，システム資源を効果的に活用する方法が普通になっている．この場合，システムに備わっている複資源は，同時に実行されている数個のタスクに動的に割り振られ，共同使用される．その結果，**図8-2**に示しているようにコンピュータ・システム全体のかなりの資源を長時間にわたって生産的なオペレーションに参加させることができる．この共同使用は，単にハードウェア資源に限らず，プログラム資源やデータ資源でも実現できるのである．

まず，**多重タスク**（multi-task）の状況のもとでのプログラムの共同使用から検討してみよう．プログラムの共同使用においては，複数個のタスクを実行するために1個のプログラム（多くの場合，特定のプログラム部分）を共有し，別々の異った処理結果を出力する（**図8-3**）．たとえば，あるデータ・セットを処理するために CPU でプログラムを実行しているさいに割り込みが生じ，その結果別のデータ・セットの処理に切りかわって同じプログラムで異っ

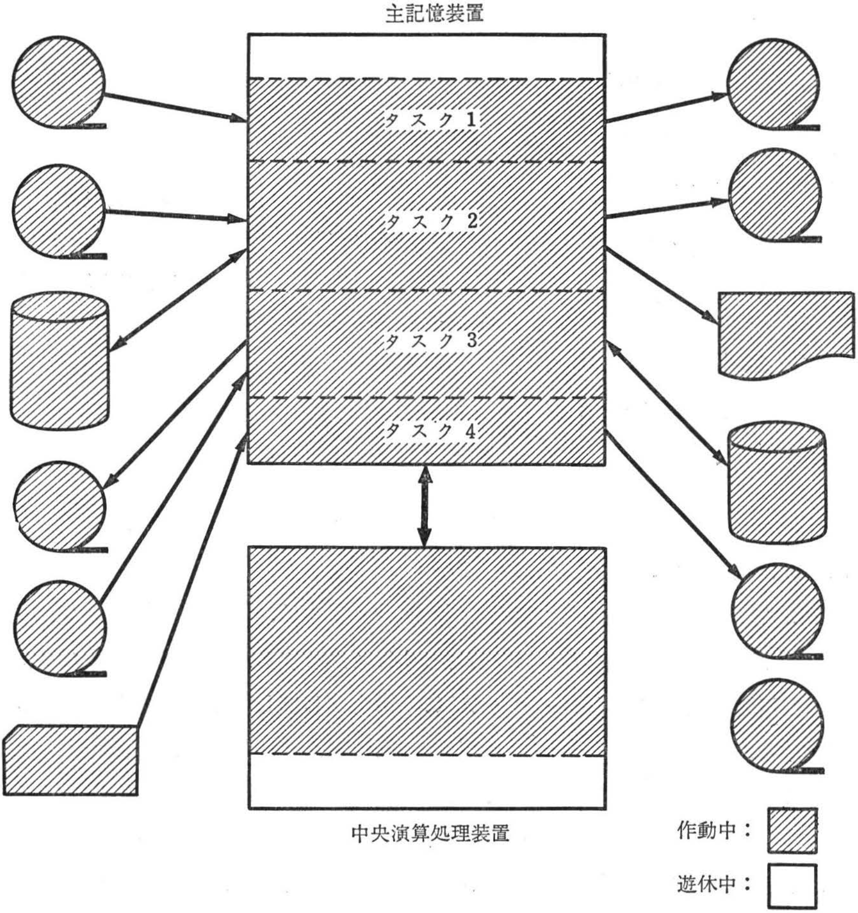


図 8-2 多重タスク・システム

たデータ・セットを処理するような場合である．このような切りかわりをくり返しながら，それぞれ別々の結果を出力してゆく．この場合には，同一プログラムを使用しながらも，異ったタスクを実行しているのである．当然のことながら，共同使用されるプログラムの作成には特別の配慮が必要である．つまり，データの処理中にプログラムが変更されるようなものであったら，別のタスクの処理結果をくるわせてしまう．したがって，この場合のプログラムは**再入可能**（reentrant）でなければならない．換言するなら，このプログラムがCPU で実行されるとき，どのような変更も加えられないように設計されてい

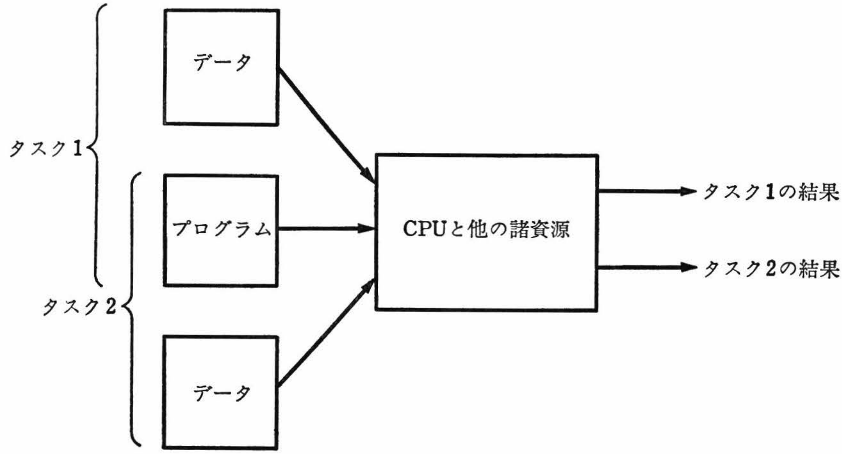


図 8-3　プログラムの共同利用

なければならない．

ところで，共同使用のできる再入可能プログラムは，IPL 時に主記憶装置にローディングし，そのまま常駐させておくことができる．制御プログラムのうち頻繁に共同利用されるルーチンにはこの方法がしばしば適用される．こうしておけば，使用のつどローディングし直す必要がなくなるので，それだけ処理効率の向上が期待できるのである．これ以外の再入可能プログラムは，特定のタスクを実行する必要が生じたときに，主記憶装置の領域にローディングする．この方法だと，主記憶装置のスペースの節約ができる．いずれにしろ，再入可能プログラムは，主記憶装置に入っているかぎり，同時に複数個のタスクで共用できるプログラムである．

一方，多重タスクを行うときには，プログラムと同様にデータも同時に共同使用することができる．**図 8-4** で示しているように，1組の共通データを使用する数個の異ったプログラムを実行して，それぞれ独立した結果を作り出すことができる．たとえば，同じデータ・セットをもとにして2種類の報告書を作るために，2つの別々のプログラムを同時に実行させるような場合である．このようなデータ・セットの共同利用を行うためには，同時に実行するいずれのタスクでもデータ・セットの内容を変更することがないことが前提となる．一

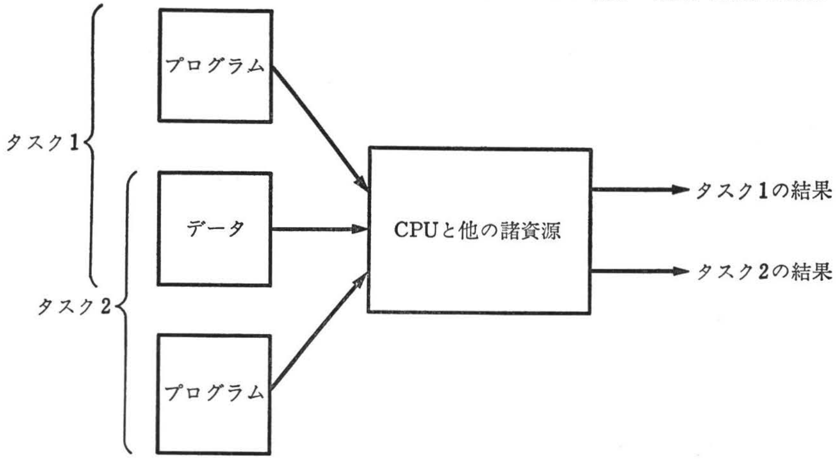


図 8-4　データの共同利用

方のタスクでデータ・セットの内容を変更しながら，他方のタスクでそれを利用して処理をしていったのでは正しい結果の保障はできなくなってしまうからである．

第7章でみてきたようにタスクの attach をするのはジョブ・スケジューラーである．しかし，タスクの実行の制御と指揮は監視プログラムの役割である．監視プログラムは，実行が開始された全てのタスクの状況を絶えず記憶し，それらの相対的な優先順位にもとづいて各タスクの実行順序を定め，タスクの同時実行を制御してゆくのである．さらに，監視プログラムはシステム内の使用可能な全資源の状況も記録保持し，特定タスクの実行を行なうために必要に応じて資源の割り振りをするのである．

資源の中には，いつでもすぐにタスクに割り振ることができる性質のもの（たとえば，主記憶装置内にある再入可能プログラム）と，一時点では1つのタスクだけにしか割り振ることのできない性質のもの（たとえば，CPU）が存在する．後者の場合，あるタスクに必要な特定の資源がすぐに使用できないと，必然的にそのタスクの実行は一時延期されることになる．その結果，このようなタスクは**待行列**（waiting queue）に入れられる．ある資源に対する待行列の中には複数個のタスクが入っている場合があるが，通常このような場合には優先順位にもとづいて配列される．しかし，入出力チャネル・タイムのよ

うな特定の資源を効率よく使用するためには，別の順序で配列されることもある．いずれにせよ，このような資源の待ち行列は，資源の需要変動をやわらげる作業貯蔵所の役割りを果す．また，準備が完了している作動可能なタスクの予備を手もとにおくことによって資源の遊びをなくし，できるだけ本来の性能で働かせ続けることによって，資源の効率的運用を行なうことにも貢献しているのである．資源の割り当ては，タスクの実行のためにその資源が本当に必要になるまで延期され，必要になった瞬間にはじめて割り振られ，かつその役割りを果したあとはすぐに資源を解放し，他のタスクに割り当てることができるようにしている．言い換えると，資源はつねに動的に割り振られ，使用中以外はそのタスクに拘束されないように仕組まれているのである．したがって，比較的短時間の間に特定の資源が沢山のタスクを実行するために幾度もくり返して用いることができるのである．

CPU 資源の割り当てに関しては**時分割機能**（time-slicing feature）を適用することもできる．この機能を用いることによって，グループ内のあるタスクが CPU を独占して，そのため，同じグループ内の他のタスクへの CPU 制御権の割り当てが遅れてしまうのを妨ぐことができる．このために，監視プログラムはあるグループ内のタスクに CPU の制御権を割り当てるさいに，各タスクに均等間隔の CPU 時間を割り振る．その CPU 時間を経過しても当該タスクがまだ終っていないと，監視プログラムが他のタスクに制御権を強制的に割り当てるようになっているのである．

ともあれ，各資源を動的に割り振って，複数のタスクの実行を制御できる監視プログラムの機能にはつぎのようないくつかの長所がある：

- １つのタスクを実行している間に，本来遊んでいる資源を他のタスクの実行に振り向けるので，それだけシステム全体の生産性が向上する．
- 主記憶装置のスペースや他の資源が新たに追加され，コンピュータ・システムの拡充がなされても，プログラムの手直しをする必要はなく，単に実行するジョブまたはタスクの数をふやすだけで，追加資源を効率よく使用することができる．

- 資源に対する需要の変動と 作業量の変化にシステムを 直ちに順応させることができるので，それだけシステムの柔軟性を増すことができる．これは，特にデータ処理量が大きく変動し，しかもそれの予測がむずかしいテレプロセシング業務にとって重要な意味をもっている．
- 数個のタスクを連続的に実行するのではなく，並行的に実行するのでそれだけ処理要求に対するシステムの**応答時間**（turn-around time）の短縮が期待できる．テレプロセシング業務では，処理要求に非常に短時間で応答する必要があるので，このことは特に重要なことになる．

これまで概観してきたシステム資源の共同利用と多重タスク処理システムの実現こそは，資源の効果的活用の本命であり，かつタスク管理の基本的機能でもある．このタスク管理の諸機能に関しての具体的内容に関しては，次節以降で詳述する．なお，システム資源の効果的管理のすべてをタスク管理で面倒みているわけではない．すでに説明してきたデータ管理やジョブ管理を通じてもシステム資源の管理の一部が実現されている．

二. タスク管理の機能とその基本構造

監視プログラムは，再三指摘してきたようにオペレーティング・システムを構成する制御プログラムの中核をなすものであり，タスク管理の機能を遂行する役割りを有している．つまり，数個のデータ処理用タスクを同時に実行するために必要となるハードウェア資源とプログラム資源の割り振りと制御をその主要な仕事としている．

すでに詳述してきたようにオペレーティング・システムでは，ジョブを受け取り，連続的に処理するようにスケジュールするためのジョブ管理の機能と，すべてのデータの記憶，保持およびその取り出しの面倒をみるデータ管理の機能とが，ここで説明するタスク管理の機能と有機的に組み合わさって所期の目標を達成してくれるのである．これら3つの管理機能の流れを図式化して示したのが**図 8-5** である．

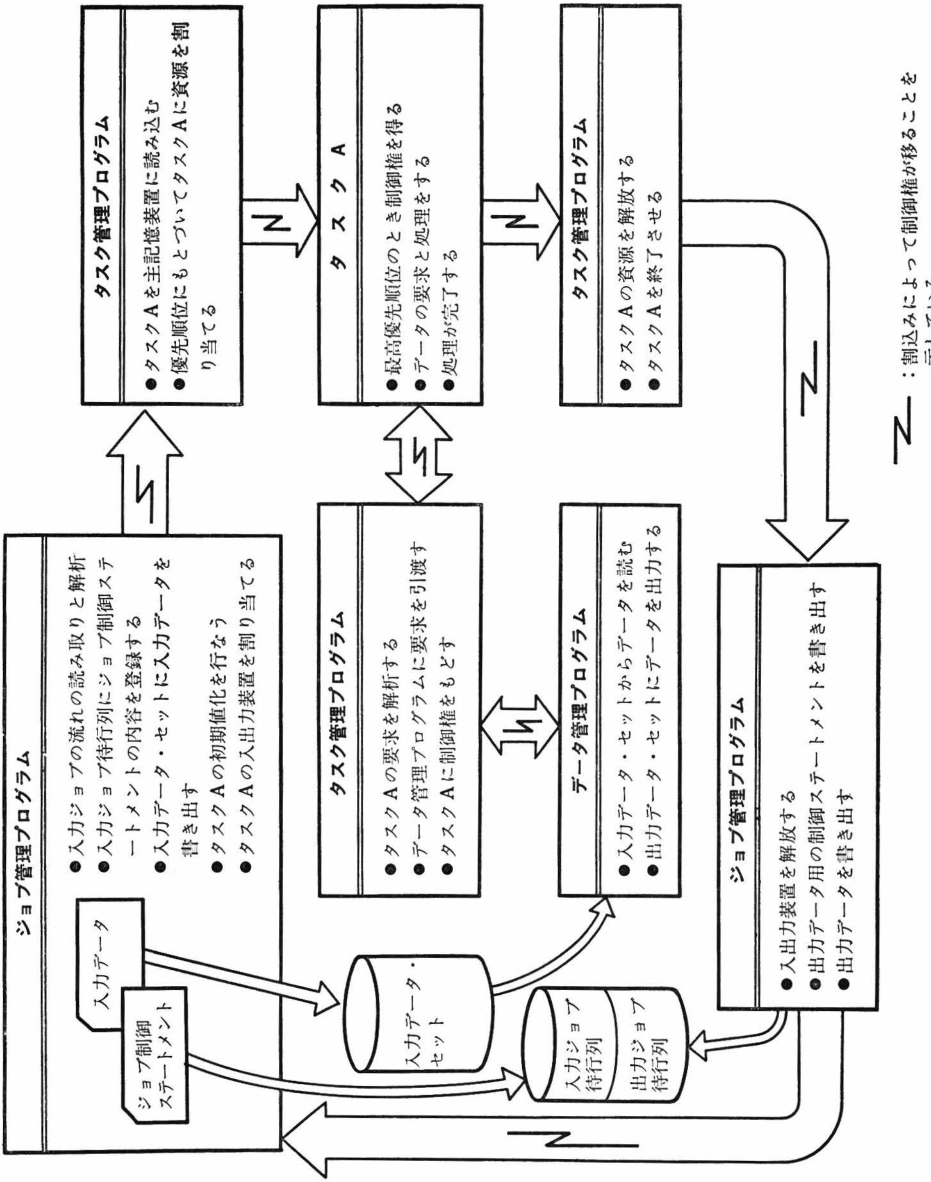


図 8-5　制御プログラムの基本的機能関係図

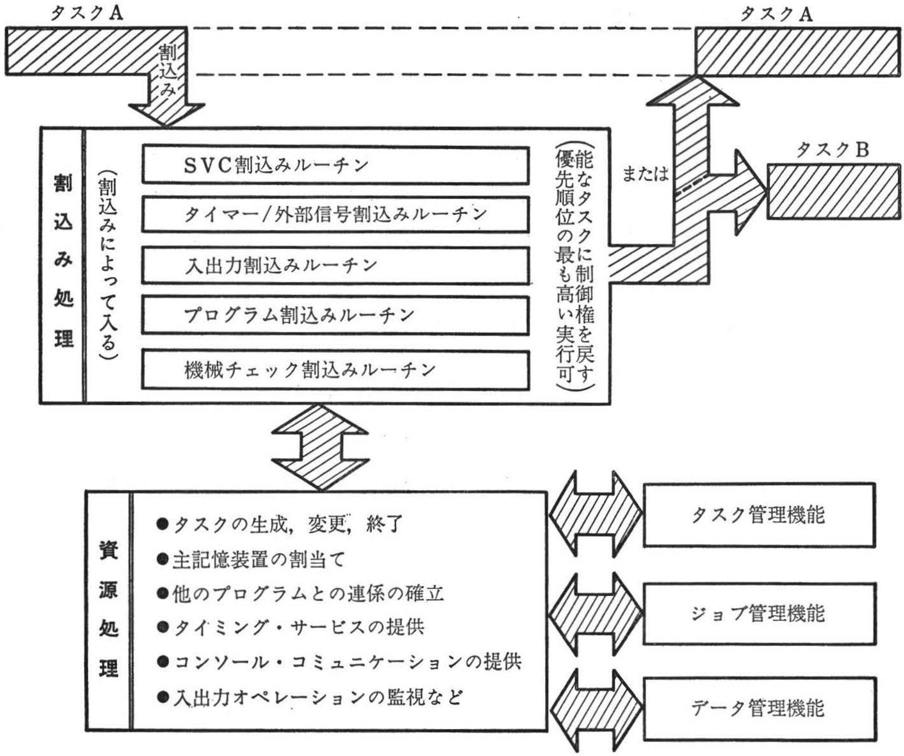


図 8-6　タスク管理機能によるサービスの処理構造

ジョブ管理ルーチンが選択したジョブ・ステップは，監視プログラムの制御のもとでは**タスク**（task）として実行される．監視プログラムは，優先順位にもとづいて必要な資源を割り振り，タスクを遂行するための資源を割り当て，これらの割り当て状況をすべて記録・保持し，タスクが完了したとき資源の解放をする．1つの資源を数個のタスクが必要とする場合には，その要求を待行列に入れる必要もある．このようにして，監視プログラムは資源の共同利用の制御を行い，CPU，主記憶装置，プログラムあるいはタイマーなどのより効率的な利用を促すのである．

このようなタスク管理の機能を果す監視プログラムは，**図 8-6** で示しているように，**割込み処理**（interruption handling）と**資源処理**（resource handling）

の2つの部分に大別できる.

1. 割込み処理と資源処理

監視プログラムは，一般には割込みを契機にして，その活動を開始する．いうまでもなく割込みは，コンピュータのハードウェア機能である．つまり，監視プログラムは，タスクを実行するに必要な資源とサービスを提供する目的で，CPU の制御権を得るための手段として，ハードウェアによる割込みを活用するのである．この割込みには，つぎの5種類が存在する：

- **SVC 割込み**：問題プログラムが監視プログラムの特別なサービスを要求するために，SVC 命令を実行することによって生じる割込み.
- **タイマー／外部信号割込み**：**間隔計時機構**（interval timer）からの注意信号，コンソール上の割込みキーないしは**直接制御機構**（direct control feature）によって引きおこされる割込み.
- **入出力割込み**：入出力に関する事象(入出力オペレーションの完了とかエラー）が生じたことによって引きおこされる割込み.
- **プログラム割込み**：プログラム上のエラーまたはデータ・エラーによって引きおこされる割込み.
- **機械チェック割込み**：機械エラーが生じたことによって引きおこされる割込み.

監視プログラムの中の**割込み処理ルーチン**（interruption handling routine）は，割込み時に使用可能となった制御情報を活用して，割込み原因を分析する．前記5つのタイプの割込みには，それぞれ2つの PSW が関係をもつ．すなわち，旧 PSW と新 PSW である．旧 PSW には，監視プログラムが割込みを分析するのに必要な情報が集約されている．一方，新 PSW にはその割込みを処理するにふさわしい割込み処理ルーチンのアドレスが示されている.

割込みが生じると，CPU は自動的に現 PSW の内容を該当割込みタイプのために用意されている旧PSW の場所に貯え，新 PSW の内容をロードする．新 PSW がロードされることによって，CPU は監視プログラム状態に切り換

わり，監視プログラムの中の割込み処理ルーチンに制御権が移される．そのあとで，監視プログラムは，割込みによって要求されているサービスを遂行するのに必要となる資源を処理する監視プログラム・ルーチンに制御権を渡すのである．

割込み処理ルーチンと資源処理ルーチンが，割込みによるサービス要求を満すためにどのように作用するかは，**図8-6**に示した通りである．監視プログラムのうちの**資源処理ルーチン**（resource-handling routine）は，つぎに示すような機能を果すシステム・プログラムから構成されている：

- **タスクの監視**（task supervision）**機能：** システムの中にどのようなタスクが存在しているか，それぞれの優先順位，使用プログラム，実行順序などを記録，保持する機能．また，サブタスクの atatch 要請に応じてタスクを生成したり，タスクの終了処理を行なったりする．
- **内容監視**（contents supervision）**機能：** 主記憶装置に存在するすべてのプログラムの記録と保持を行い，かつプログラム間の連係を行なう．また，要求されたプログラムを，データ管理機能と連係して，ローディングする機能をもっている．
- **主記憶装置監視**（main storage supervision）**機能：** 主記憶装置のスペースを割り当てたり，使用済みのスペースを解放する機能であり，主記憶装置のどの部分がどのように使われているかの記録と保持の仕事を行なう．
- **タイマー監視**（timer supervision）**機能：** タイマー用のマクロ命令で提供される情報にもとづいて間隔計時機構のセットや維持を遂行する機能である．
- **オーバレイ監視**（overlay supervision）**機能：** オーバレイの情報にもとづいて，プログラムのオーバレイに関する制御を行なう機能．詳しくは第4章「プログラム設計構造と連係編集プログラム」を参照されたい．
- **コミュニケーション・タスク機能：** オペレータがシステムと直接コミュニケートしたり，プログラムでオペレータにコンソール・メッセージを書

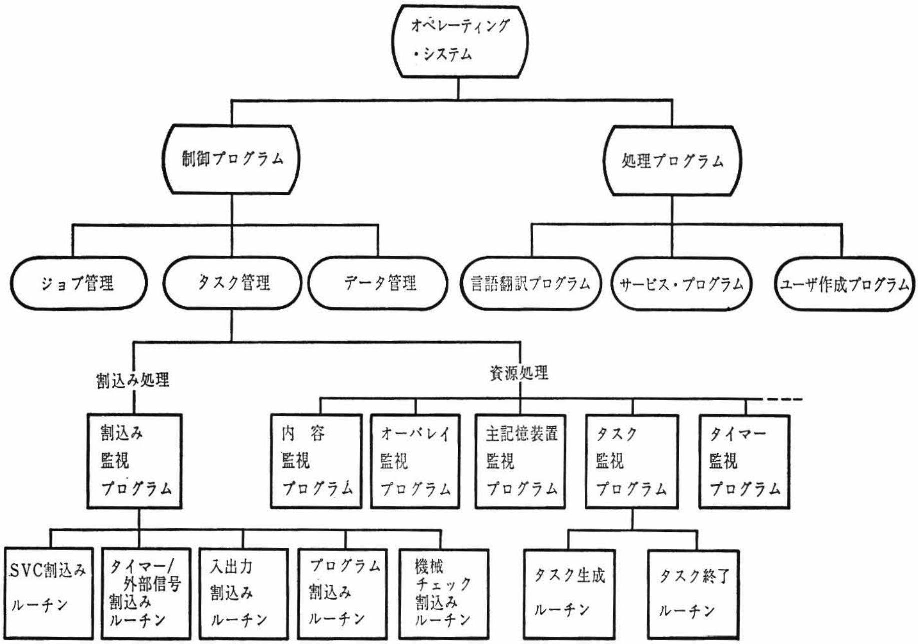


図 8-7　タスク管理機能の位置づけ

き出したりするさいの監視を行なう機能．詳しくは第7章「ジョブ管理の諸機能とその構造」の「マスター・スケジューラーの機能構造」の項を参照されたい．

このようなサービス機能を果たしたあと，監視プログラムはつぎに制御権を受け渡すべきタスクを決定し，該当タスクに制御権を渡す．

これまで説明してきたタスク管理の機能を制御プログラムと関係づけてまとめたのが**図 8-7** である．これらの機能のうち主要なものに関して以下でその機能構造を明らかにしてゆくことにする．

2.　割込み処理機能

監視プログラムに CPU の制御権が移るのは，割込みが生じたときだけである．つまり，監視プログラムの機能は割込みによってはじめて生かされるので

ある．監視プログラムは，割込み原因を分析し，その処理にふさわしい監視プログラム・ルーチンを決定する．その結果，**図 8-8** で示したような処理ルーチンが選択されることになる．

機械チェック割込み以外の割込みが生じた場合には，CPU の制御権が割込みを起したプログラムから監視プログラムの中の**割込み処理ルーチン**（interruption handling routine）に移される．機械チェック割込みを除いたそれぞれの割込みに対応した4種類の割込み処理ルーチンが用意されているのである．一方，機械チェック割込みの場合には，**回復管理ルーチン**（recovery manage-

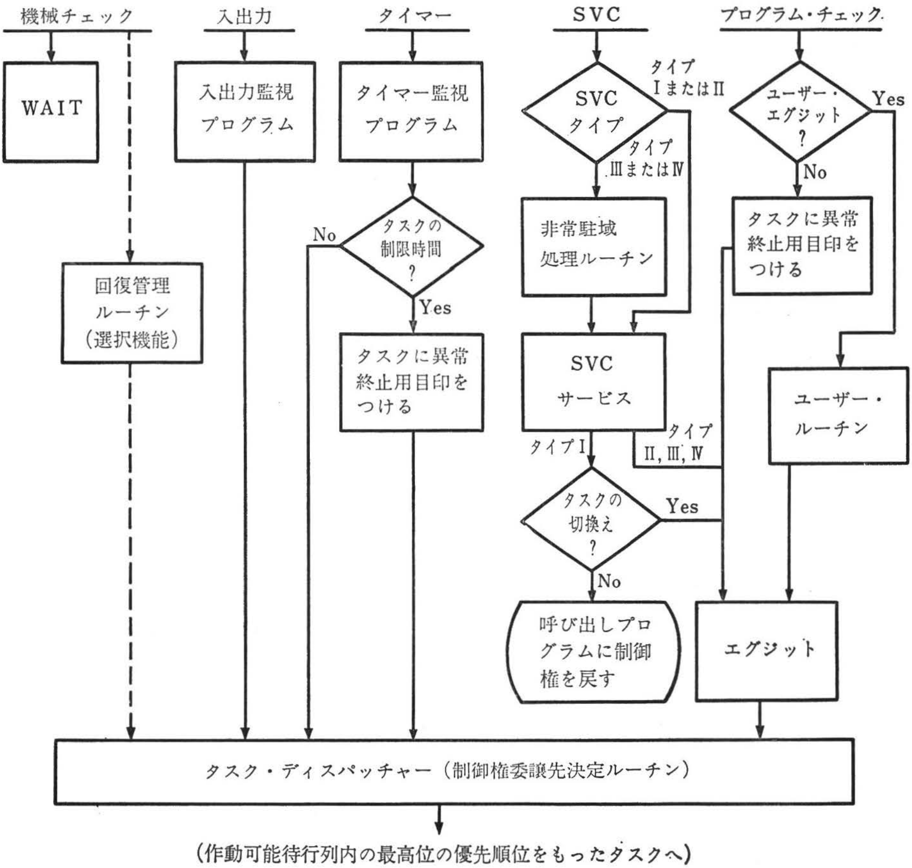


図 8-8 割込みプロセシングの機能構造

ment routine）がシステムに組み込まれていると，制御権は直接このルーチンに渡されてしまう．この回復管理ルーチンが組み込まれていないと，システムは**待ち状態**（wating state）におかれることになる．

ところで，割込み処理ルーチンは，割込み自体の処理を行うことはせず，単に割込みを分析し，それに見合った**割込みプロセシング・ルーチン**（interruption processing routine）に制御権を渡すだけである．つまり，SVC 割込み処理ルーチンは，SVC 命令の中で示されている SVC ルーチンを必要に応じてローディングし，そのルーチンに制御権を渡すだけである．入出力割込み処理ルーチンは，入出力監視プログラムに制御権を移す役割りを果し，タイマー／外部信号割込みルーチンは，何が原因で割込みが生じたのか分析しそれに適したルーチンに制御権を渡すのである．同様に，プログラム割込み処理ルーチンでは，その割込みを処理するためのユーザー・ルーチンが組み込まれているかどうかをチェックし，組み込まれている場合にはそのルーチンに制御権を移すし，そうでない場合には割込みを起したプログラムで実行されているタスクを終らせてしまうルーチンに制御権を渡す．なお，プログラム割込みを起したプログラムが監視プログラム状態にある場合には，関連するタスクが何時でも**異常終止**（abnormal end）になってしまう．

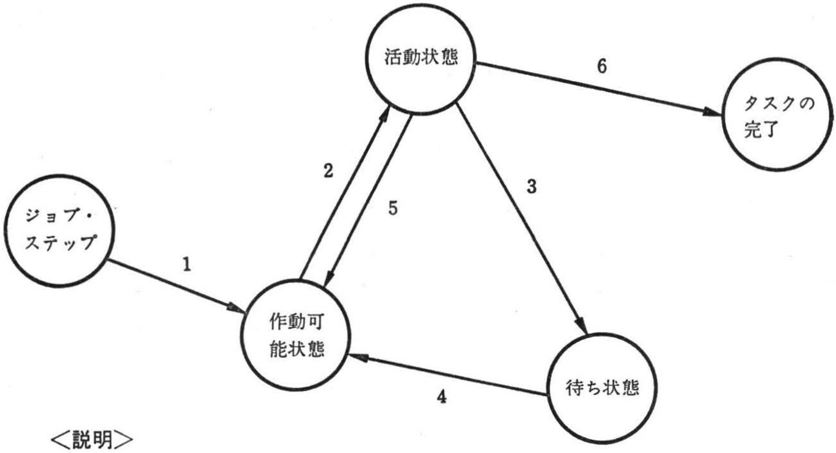
いずれにしろ，割込み自体の処理を受け持つのは**割込み処理ルーチン**（interruption handling routine）ではなく，**割込みプロセシング・ルーチン**（interruption processing routine）なのである．この様子を示したのが図 8-8 である．

割込みを処理している間は，割込み処理ルーチンが機械チェック割込みを除いたすべての割込みを禁止または延期状態におくのが普通であり，割込みを起したプログラムに関する重要な情報を完全に保管しておくことが可能となる．この重要な情報とは，割込み処理が終ったあと再び割込みを起したプログラムに制御権を戻すのに必要となる汎用レジスターの内容とか PSW の情報のことである．ともあれ，割込み処理が終ると，**タスク・ディスパッチャ**（task dispatcher）は，**作動可能待ち行列**（ready queue）で待たされているタスクのうち最も優先順位の高いタスクの処理プログラムの中断点に制御権を戻すのであ

る.

3. タスクの監視機能

タスク監視プログラムは，必要なタスクをシステム内に作りあげ，それらタスクの実行過程を監視し，監視的サービスが終るつど，タスクを遂行するプログラムに制御権を移す．また，タスクの実行が完了したときには，その後始末的な処理の面倒をみる．システム内で実行されるタスクは，タスク制御ブロック（Task Control Block）で代表される．したがって，タスク監視プログラムは，この TCB をタスクの状態に合わせて修正し，TCB 待行列を保持してゆく



＜説明＞

① ジョブ・ステップがタスクとして attach され，タスク制御ブロックが作動可能待行列に入れられる.

② 該当作動可能タスクが他の作動可能タスクのどれよりも高い優先順位を持っていると， CPU の制御権が渡されそのタスクが実行される．これが，**活動状態**（active state）である.

③ 何らかの事象が完了するのを待つために，タスクは**待ち状態**（wait state）におかれる.

④ 待っている事象が完了すると，そのタスクは**作動可能状態**（ready state）に移る.

⑤ 活動中のタスクよりも優制順位の高いタスクが作動可能になることによって，そのタスクは CPU 制御権を譲渡し再び作動可能状態に戻される.

⑥ タスクの実行が完了すると，タスク制御ブロックは作動可能待行列から削除され，そのタスクが使用していた各種資源は解放され，システムで利用することができるようになる.

図 8-9 タスクの状態

ことによって，前述の仕事を達成してくれるのである．**図8-9**で示しているように，タスクは**作動可能状態**（ready state), **活動状態**（active state）および**待ち状態**（wait state）の3つの状態をくり返しながら実行されるのである．これらの状態を TCB を主体に管理してゆくのがタスク監視の機能といえる．

ジョブ・スケジューラーがジョブ・ステップの初期化作業を行ったあとで，ATTACH マクロ命令を出すことによって，監視プログラムはそのステップ用タスクを作りあげる．ジョブ・ステップは，人間が定めた作業単位であり，ジョブ・スケジューラーによって処理される単位であるのに対して，タスクは監視プログラムが資源の割り振りをし，制御してゆく単位である．つまり，タスクはシステムの中での制御単位といえるのである．ともあれ，監視プログラムは，このジョブ・ステップ・タスクと TCB およびそれに関連した制御ブロックやテーブルを活用することによって，もともとはジョブ・ステップとして指定されたタスクの処理過程を制御してゆくのである．なお，ATTACH マクロ命令が機械語命令になると，その中には SVC 命令が含まれており，これで SVC 割込みを引きおこすようになっている．したがって，CPU の制御権は，SVC 割込み処理ルーチンを通じて，**タスクを取りつける SVC ルーチン**(attach SVC routine）に渡される．このルーチンで TCB を作りあげ，さらにこのタスクで最初に使用するプログラムのための**要求ブロック**(Request Block; 略して **RB**）を生成してくれるのである．これらブロックは，それぞれ **TCB 待ち行列**（TCB queue）および **RB 待ち行列**（RB queue）に入れられる．

ところで，制御プログラムでタスクに関する情報が必要になると，必ずTCB待行列から捜し始めるようになっている．言うまでもなく，システム内で同時に実行されているすべてのタスクに対応して1つずつ TCB が用意されており，この TCB がそれぞれのタスクの状態や特性を示しているのである．しかも，タスクの状態を示すほとんどの情報は，関連する別の制御ブロックや待行列のポインターとそのタスクを実行するに当って必要となる資源に関する制御情報から構成されている．TCB は該当タスクの優先順位にもとづいて待行列に入れられ，その待行列の中でつぎの優先順位にあるタスクの TCB をポイントする仕組みになっている．なお，最初の TCB だけは，**中核プログラム**

(nucleus) の中にある**コミュニケーション・ベクトル・テーブル** (Communication Vector Table; 略して **CVT**) でポイントされている．したがって，制御プログラムは CVT を参照することによって，すべての TCB をつきとめてゆくことが可能となる．また，TCB の中に示されていてポインターを用いれば関連する制御ブロックやテーブルから情報を得ることができるのであり，いわば TCB はタスクに関する全情報の中継的役割をも果しているのである．

一方，RB は該当タスクのプログラムで呼び出されたプログラムそれぞれに対応して作られる．LINK, XCTL あるいは ATTACH マクロ命令のような監視プログラムの援助によって連係されるプログラムであるとか，プログラムで呼び出されるタイプⅡ，Ⅲ，Ⅳの SVC ルーチンのそれぞれに対して RB が作られるのである．RB は RB 待行列に入れられるが，この RB 待行列も TCB からポイントされている．ともあれ，この RB 待行列をみれば該当タスクを実行するのに用いられているプログラムがすぐに判別できるのである．

タスクの実行中に新らしいプログラムが呼び出されて待行列に RB が追加されたり，逆にそれらのプログラムの使用が終ったことにより待行列から削除されることによって，監視プログラムはどの時点でどのプログラムに制御権を受渡すべきかを決定することが可能となる．あるプログラムの実行が完了すると，待行列では関連タスクが完全に終了したのかあるいは別のプログラムを実行する必要があるのかどうかを示してくれる．つまり，終了したプログラムの RB だけが待行列に残されているのであれば，そのタスクは完了したことを意味するし，別の RB が残されているのなら，これら RB で示されているプログラムをさらに実行しないことにはタスクは完了しないことを意味しているのである．

ともあれ，あるタスクに関連する最後のプログラムの実行が完了すると，タスク監視プログラムは TCB 待行列から該当 TCB を削除する．もし，完了したタスクがサブタスクではなくてジョブ・ステップ用タスクであると，ジョブ・スケジューラーの中の終止プログラムが働き，そのジョブ・ステップに関する後始末的処理を行って，ジョブ・ステップの完了とする．

このようにして，システム内に存在するタスクとそれぞれの優先順位にもと

づいた管理を TCB 待行列を用いて行ない，かつ使用プログラムとその実行順序の制御を RB 待行列を用いて行なうのである．

さて，タスク監視プログラムが割込みに対するサービスをしたあと，どのような方法で CPU の制御権をタスク・プログラムに受け渡すのかを検討しておこう．

監視プログラムが割込み処理を完了すると，当然のことながら TCB の制御のもとで作動しているプログラムに CPU の制御権を戻す．制御権を受け取るプログラムが最後に割込みを起したプログラムであることが最初から決定している場合だと，CPU 制御権は直接それに戻されてしまう．このようなケースはまれであり，タイプＩのある種の SVC ルーチンが割込み処理をした場合に生じるだけである（図 8-8 の SVC 割込みの個所を参照）．多くの場合には，割込みを起したプログラム以外のプログラムが制御権を受け取ることも可能であり，このような場合には制御権はいったん**タスク・ディスパッチヤ**（task dispatcher または dispatcher routine）に渡される．

タスク・ディスパッチャは TCB 待行列をチェックし，最も優先順位が高くかつ作動可能状態のタスクを選び出し，そのタスクを遂行するために目下使用中のプログラムに制御権を渡す．したがって，指名されるタスクは必ずしも最後に割込みを引きおこしたタスクとは限らない（図 8-10 参照）．割込み処理を行うことによって，より優先順位の高いタスクを作動可能状態にしたり，新しいタスクを生成したりすることがあるので，割込みを起したプログラム以外に制御権が移ることも考えられるのである．

同じ優先順位をもった数個のタスクが作動可能状態にあると，TCB 待行列の中で下部に位置づけられているタスクは上位にあるタスクが完了するか，待ち状態にならない限り，CPU の制御権は渡ってこないのが普通である．この種のタスクにも CPU 制御権を平等に渡すようにするのが，前述した**時分割機能**（time-slicing feature）である．

時分割とは，特定優先順位を持ったタスク（これを**時分割グループ**という）のそれぞれに，あるタスクが一定時間経過しても何らかの理由で制御権を受け渡さないときに，強制的に制御権を譲渡させてしまう機能である．通常は，タ

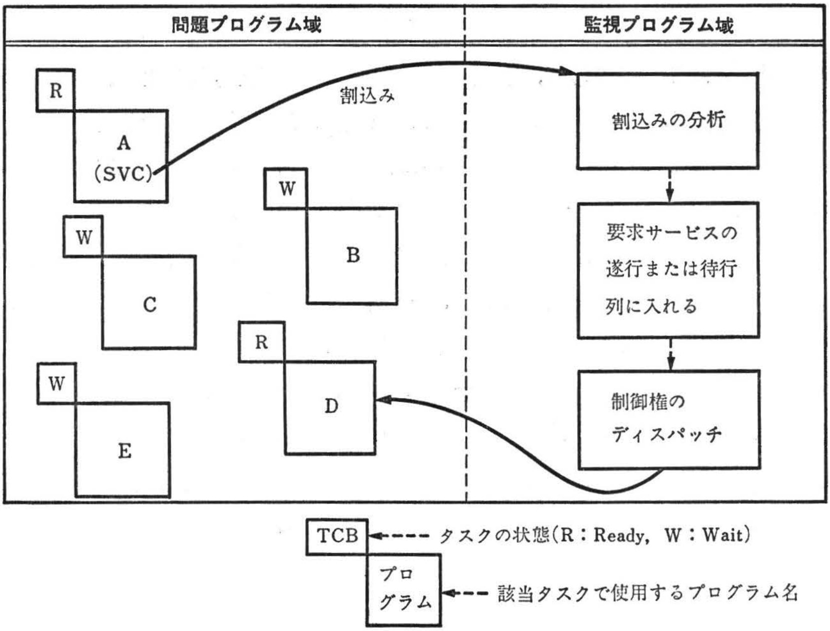


図 8-10 割込処理とタスク・ディスパッチング

スクが完了するか，より優先順位の高いタスクが作動可能状態になるか，あるいはそのタスクが入出力オペレーションのような特別な事象によって待ち状態にならない限り，制御権を保持し続ける．ところで，時分割タスクの場合には，これら3つの条件のいずれかが生じるか，または設定された時間間隔が過ぎるまで制御権を保持する．設定された時間が過ぎるとタイマー割込みが生じ，時分割グループ内の次の作動可能タスクに制御権が渡される．TCB 待行列内での TCB 配置の順番でグループ内のそれぞれの作動可能タスクに制御権が移る仕組みになっているのである．なお，上位の優先順位をもったタスクが作動可能状態になったり，時分割グループ内の全タスクが完了するか待ち状態になった場合には時分割機能は働かなくなる．また，タスクが待ち状態になったり，優先順位の高いタスクが作動可能となったために時分割タスクが制御権を失った場合には，指定時間間隔の残りの部分は無視されてしまう．つまり，再び制御権が時分割グループに戻ってきたとしても，制御権を失ったタスクではなく次のタスクに制御権が与えられることになる．

時分割グループや時分割の時間間隔はオペレーティング・システムの生成時に指示する．これによって，中核プログラムが初期値化されるとき，**時分割制御エレメント**(Time-Slice Control Element; 略して **TSCE**) がそれぞれの時分割グループごとに中核プログラムの中に作り出されるのである．この TSCE には，その時分割グループの時間間隔およびグループ内の最初の TCB，最後の TCB，さらにつぎに制御権を与えるべきタスクのの TCB それぞれのポインターが収容できるようになっている．一方，TCB では該当タスクが時分割グループのメンバーであるかどうかを示している．最も優先順位が高くて作動可能状態にあるタスクが時分割グループに属しているときは，タスク・ディスパッチャはTSCE をチェックし，そのグループの次のタスクに制御権を渡すようにしているのである．これによって，同じタスクがくり返し制御権を占有してしまうことがないようにすることができるのである．

4.　主記憶装置の監視機能

主記憶装置の監視ルーチンは，**動的記憶域** (dynamic storage area) と**システム待合せ域** (system queue area) の制御と割当ての機能を果す．動的記憶域はジョブ・ステップとシステム・タスクに割り当てられ，システム待合せ域は監視プログラムが利用する各種制御ブロックを収容する場所として割り振られる．

まずは，動的記憶域がどのように割り振られてゆくかを検討しよう．

もともと動的記憶域は，未割当ての連続した主記憶域のブロックの集まりから構成されたものである．ここでいうブロックとは2048バイト，つまり2K単位の記憶域のことである．記憶保護の単位が2Kであることから，監視プログラムが主記憶装置の割当てや制御を行なうさいにも必然的にこの2K単位が基準になる．なお，オペレーティング・システムのもとでの主記憶装置の一般的構成は**図 8-11** に示した通りである．

ともあれ，システム・タスクないしはジョブ・ステップが開始されると，開始ルーチンでそのタスクなりステップなりが必要とする主記憶域を要求する．その結果，**主記憶装置監視プログラム** (main storage supervisor) は，動的記

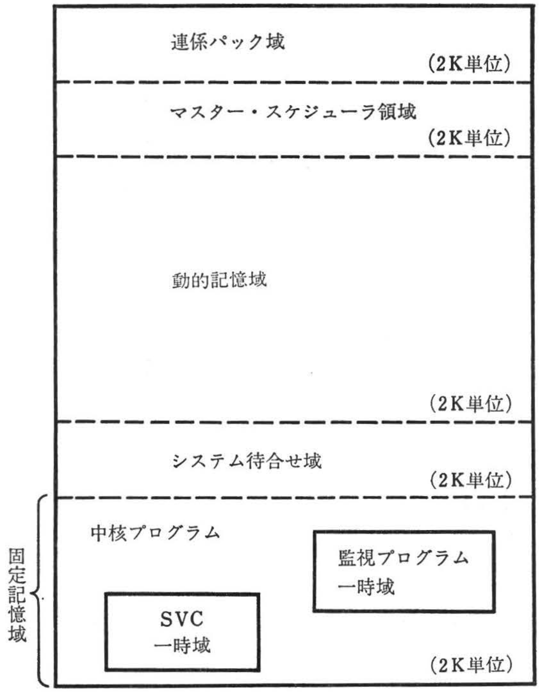


図 8-11　主記憶装置の一般的構成

憶域のアドレスの上位の方から必要な大きさだけ割り当てる．割り当てる記憶域は，連続した記憶域でブロック単位の大きさである．このようにしてシステム・タスクないしはジョブ・ステップに割り振られた2K単位のブロックの集まりが，**領域**（region）である．領域内の記憶域は，その領域で実行されるプログラムの要求に応じて割り振られてゆく．なお，**図 8-12** で示しているように領域内の記憶域は，**PCI**（Program Controlled Interruption）呼び出し用の作業域，レジスターやパラメータ情報を保存する記憶域および連係パック域にロードしていない再入可能ルーチンのための記憶域がとられる．したがって，処理プログラムが GETMAIN マクロ命令などを通じて要求する記憶域は，領域の中の未割当ての部分（フリー・スペース）から割り当てられるのである．

通常は，該当ステップやタスクから要求される全記憶域をその領域から割り当てる．しかし，**ロールアウト／ロールイン機能**（rollout/rollin feature）を適

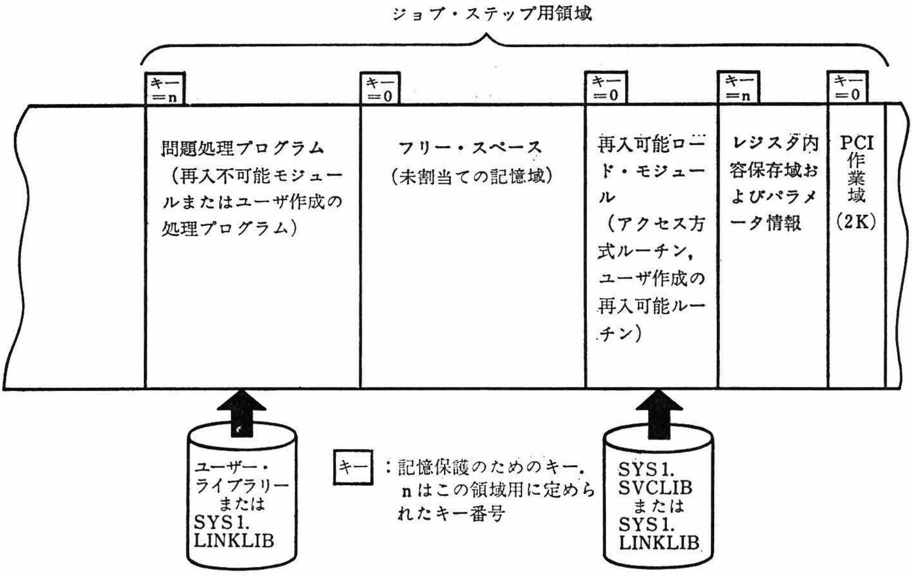


図 8-12　処理プログラムの領域の構成

用すると，そのステップ用の領域だけで満すことのできない要求を追加の領域を一時的に割り振ることによって満足させることができる．このような余計の記憶域が必要になると，ロールアウト／ロールインのルーチンは，まず動的記憶域のうちまだ領域に割り振っていない未使用の場所からそのステップのために新規の領域を割り当てようと試みる．不幸にして，未使用の記憶域が十分ないと，別のジョブ・ステップに割当てずみの領域の内容をそのまま直接アクセス・ボリューム上に書き出し（このことを**ロールアウト**と言うのだが），その領域を余分のスペースを必要とするステップに使わせるのである．追加の領域の使用が終ると，ロールアウトした内容をその領域に呼び戻し（これが**ロールイン**である），そのステップのオペレーションを続行させる．

ところで，主記憶装置監視プログラムは，記憶装置のどの部分が未使用で，どの部分が使用中であるかの判定をどのようにして行なうのであろうか．このために，**区分待行列要素**（Partition Queue Element; 略して **PQE**）と**フリー・ブロック待行列要素**（Free Block Queue Element; 略して **FBQE**）と呼ばれる制御ブロックを用いる．該当領域の PQE は，その領域が割り当てられたとき

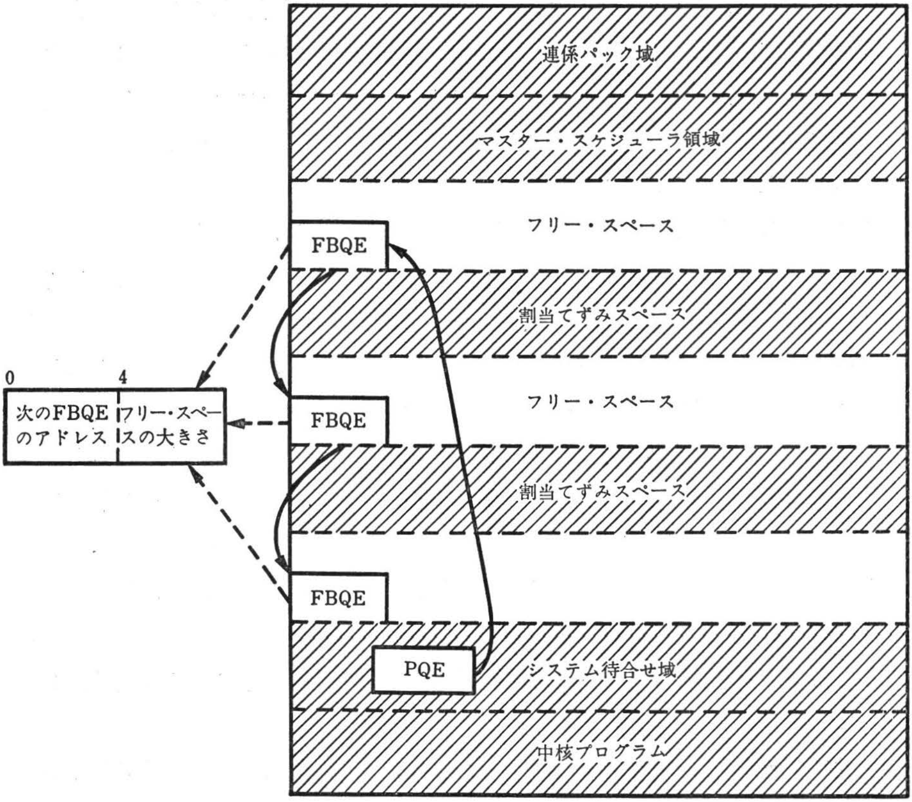


図 8-13　未割当てスペースの探索構造

にシステム待合せ域の中に作りあげられる．PQE は，その領域内にあるすべてのタスクの TCB でポイントされる．一方，PQE 自体は FBQE で構成される待行列の最初のものを指すようになっている（**図 8-13** 参照）．FBQE は，領域内の連続した**フリー・スペース** (free space) のブロックの集まりごとに作成され，最初のブロックの下位アドレス8バイトに収容される．したがって，PQE を起点として，それぞれの FBQE を調べてゆくと，どこにどれだけの利用可能なスペースがあるかが判明するのである．

領域の中の全記憶域が利用可能であるとか，利用可能な記憶域のすべてが連続している場合には，唯一つの FBQE が存在するだけである．一方，記憶域が

割り当てられ，それが利用され，そのあとで使用解除が行われるといったサイクルが何回かくり返されると，領域の中のあちこちに未割当ての区域が散在するようになる．このような記憶装置の断片化が生じることによって，各々の未割当て区域に対応する FBQE が作られ，待行列を形式することになる（図8-13を参照）．しかし，ある記憶域の使用解除がなされることによって，いくつかの区域から大きな使用可能域が形作られる場合もある．このような場合には不要な FBQE は待行列から削除され，新しく形成された大きな区域に対して1個の FBQE ですむようになる．

一方，記憶域の要求があると，FBQE を探索してゆき，その要求を満すに十分な区域を見つけ出す．万一その要求を満すに足る大きさの連続した区域がなく，かつロールアウト機能も用いていない場合には，そのタスクは異常終止せざるを得ない．逆に，要求を満足する場合だと，未割当ての記憶域から十分な数の記憶ブロックが割り当てられ，**サブプール**（subpool）の一部となる．なお，どのサブプールに割り当てられるかは，その記憶域がどのように使われるかにかかっている．

ここで，サブプールの考え方と記憶域のサブプールへの割り振りの様子について言及しておこう．

サブプールとは，**サブプール番号**（subpool number）と呼ばれる目印しをつけて特定のタスクに割り当てたすべての2Kブロック単位の記憶域のことである．ある領域が確保された時点には，まだその記憶域の割り振りは全く行なわれておらず，したがって記憶保護用のキーはゼロであるし，サブプールも存在しない．しかし，領域内の記憶域が要求されると，その領域内の未割当ての記憶域がサブプールの一部となる．記憶域の要求にさいしては，その記憶域をどのサブプールに割り振るかをサブプール番号で指定する（ただし，後述するように必ずしも明示的に指示する必要はない．省略時の解釈がなされる場合がある）．

特定タスクの特定サブプールを指定した最初の要求が出されたときには，いまだサブプールが存在していないので，領域の中の未割当ての記憶域から，十分な個数の連続したブロックを割り振ることによってサブプールを作りあげて

くれる．一方，前の要求によってすでに指定したサブプールが存在している場合だと，そのサブプールを構成している記憶域をチェックし，要求を満すに十分な連続した記憶域が利用可能であるかどうかを調べる．利用可能であれば，その記憶域が要求を満すのに使われる．逆の場合には，その領域の中の残っているフリー・スペースから十分な個数の連続したブロックを探し出し，そのサブプールに割り振る．その領域に十分なスペースが残されておらず，かつロールイン／ロールアウト機能が適用されないなら，当然のことながらそのタスクは異常終止してしまう．ともあれ，サブプール内の記憶域を要求するのは，タスクを遂行する処理プログラムの場合もあるし，制御プログラムの場合もある．前者の場合には作業域として，後者の場合にはプログラムをロードする区域または作業域としての要求である．

サブプールは，通常特定タスクに割り振られる．ところで，ある領域が複数個のタスクで使われている場合，つまり，ジョブ・ステップ・タスクがサブタスクを attach したようなときには，それぞれのタスクが番号 0 から 127 を持った別々のサブプールを有することもできるし，0 から 127 のサブプールをタスク間で共有することもできる．なお，どんな領域にもサブプール 251 と 252 が必須となる．

監視プログラム状態で働く制御プログラムがある領域内の記憶域を必要とするさいには，サブプール 251 ないしは 252 の記憶域を要求するのが普通である．この場合，サブプール 252 は SYS 1. SVCLIB または SYS 1. LINKLIB から呼び出した再入可能ルーチンを収容するための記憶域か，記憶保護を必要とする制御プログラム用データを収容する記憶域のいずれかに適用される（**図 8-14** の一番上の図を参照のこと）．サブプール 252 に割り振られる記憶域にはゼロの記憶保護キーが与えられるので，通常の処理プログラムでサブプール 252 の記憶域に書き込みを行うことは禁止される．なお，サブプール 252 に割り振られる記憶域は該当領域の上位アドレスの記憶域から確保されていく．

一方，サブプール 251 は**逐次再使用可能プログラム** (serially reusable program)，**再使用不可能プログラム** (non-reusable program) およびユーザー・ライブラリーにある再入可能プログラムなどを収容するのに必要な記憶域に適用

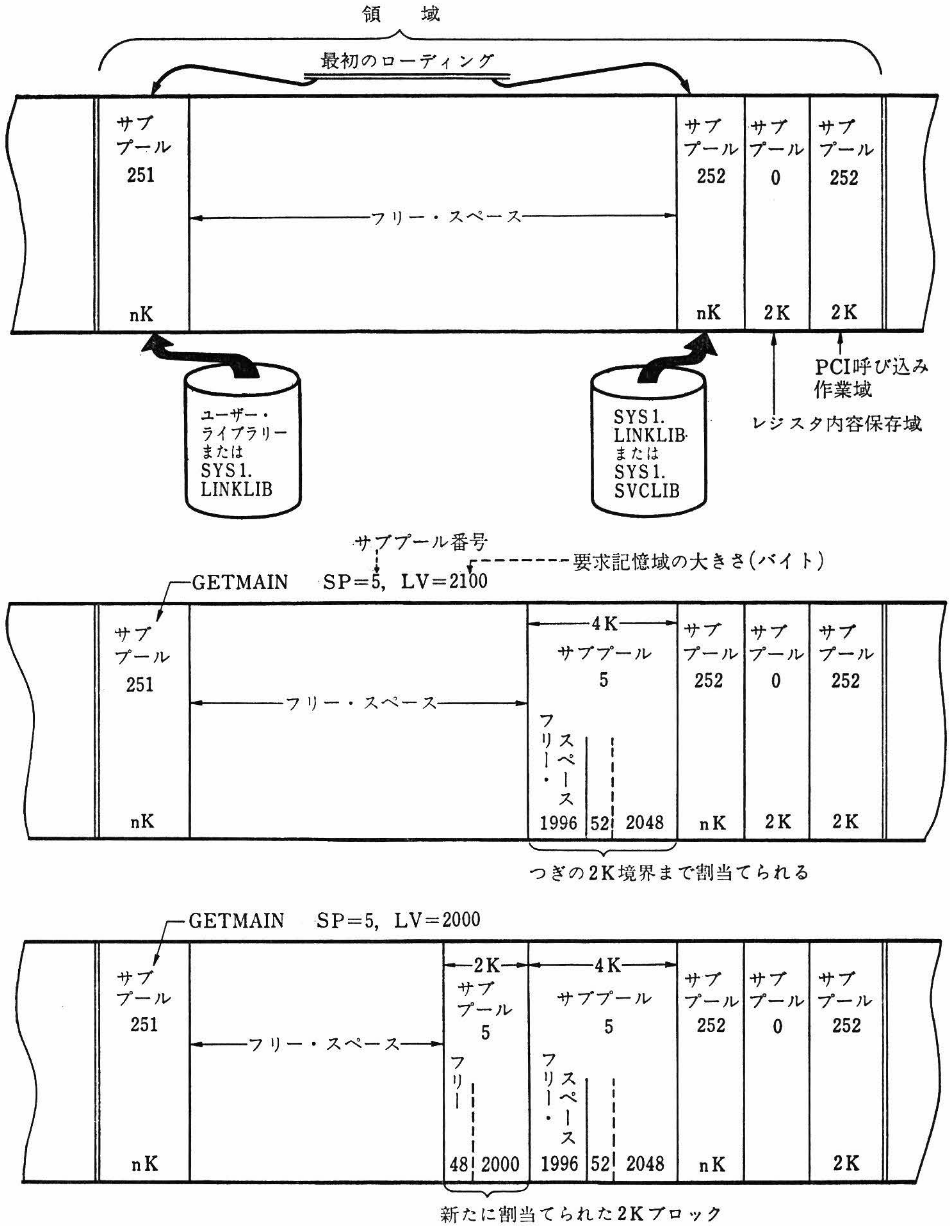


図 8-14　サブプール割当ての様子

される（**図 8-14** の一番上の図を参照のこと）. サブプール 251 は該当領域内の下位アドレスの記憶域から確保されてゆく. サブプール 251 の記憶域には，その領域を使用するプログラムと同じ記憶保護キーが与えられるので，処理プログラムでその内容を変更することができる. なお，該当領域のサブプール 251 と252 を総称して**ジョブ・パック域**（job pack area）と呼ぶこともある.

図 8-14 に示しているように，領域が確保されプログラムがローディングされる最初の段階に，主記憶装置監視プログラムはサブプール 251 と 252 を割り当て，所要のプログラム・ルーチンをローディングしてくれる. さらに，タスクの遂行中に処理プログラムで GETMAIN ないし GETPOOL マクロ命令を通じて記憶域の要求を行なうと，指定されたサブプールから記憶域が確保されるのである. なお，サブプールはいつでも 2 K 単位で割り振られてゆく.

ところで，新しいサブプールが作られるごとに，そのサブプールに対応した**サブプール待ち行列要素**（Subpool Queue Element; 略して **SPQE**）が該当タスクのサブプール待行列に入れられる. このサブプール待行列は，一連の SPQE から構成されており，当然のことながらその場所は TCB からポイントされている. 主記憶装置監視プログラムは，このサブプール待行列を参照することによって，どのサブプールがタスクで使用されており，どの記憶域を各々のサブプールに割り振るべきかを判定するのである.

最後に，**システム待合せ域**（system queue area）の記憶域の割り振りについてふれておこう.

システム待合せ域は，制御プログラムだけが変更することのできる各種制御ブロックを格納するための記憶域として使用される. したがって，この記憶域の記憶保護キーはゼロであり，キーがゼロのもとで作動するプログラムだけがその内容を書きかえることができるのである.

システム待合せ域に対する記憶域の要求を満すことができない場合には，隣接する動的記憶域を 2 K ブロック単位で割り振ることによって，待合せ域を拡張することが可能である. ただし，監視プログラムは，このような拡張を行なうまえに，まず各領域内の使用済みのスペースを解除する. いくつかのスペースが解除できると，それらを反映したシステム待合せ域内のスペース制御用の

ブロックが削除可能となるからだ．この制御用ブロックの削除によって，記憶域の要求量を満すに足るシステム待ち合せ域が得られるなら，拡張操作は行なわなくてすむのである．制御用ブロックの削除をしても十分なスペースが確保できないとなると，動的記憶域から余分のスペースを確保し，システム待合せ域を拡張するのである．

いったんシステム待合せ域が拡張されたとなると，その区域はIPL手順がとられないかぎりもとのサイズにもどすことはできない．なお，システム待合せ域に隣接する動的記憶域がすでに領域として割り振られてしまっていると，システム待合せ域を拡張することはできず，結局，システムは**待ち状態**（wait state)におちいってしまうのである．

以上のように動的記憶域とシステム待合せ域の割り振りと制御の機能を担当するのが，**主記憶装置監視プログラム**（main storage supervisor）である．

5.　内容監視の機能

内容監視プログラム（contents supervisor）は，非常駐ルーチンの主記憶装置への呼び込みと，動的記憶域と**連係パック域**（link pack area）にどのようなルーチンが存在しているかの記録を保持する役割りを果す．非常駐ルーチンは，第4章「プログラム設計構造と連係編集プログラム」の個所で詳述したようにLINK, ATTACH, XCTLおよびLOADマクロ命令が実行された結果として呼び込まれるものである．ただし，該当ルーチンがすでに主記憶域に存在している場合だと，そのルーチンの属性と記憶されている場所によって，呼び出しルーチンでそのまま利用することも可能である．これをコントロールするためには，どのようなルーチンが主記憶装置のどこに記憶されているかの情報を管理する必要がある．

呼び出されるルーチン（called routine）の属性とそれの記憶されている場所によって，**呼び出しルーチン**（calling routine）で使用可能であるかどうかの判定がなされるということについて少し説明を加えておこう．

連係パック域に収容されているルーチンは，すべて再入可能なルーチンであり，どんな呼び出しプログラムからでも使用できる．しかも，連係パック域の

1個のルーチンが同時に複数個の呼び出しルーチンで利用可能である．一方，領域に呼び込まれた再入可能ないしは逐次再使用可能ルーチンは，その領域内にあるタスクを遂行するルーチンだけから再使用できる．なお，再使用不可能なルーチンだと，たとえそのルーチンが主記憶装置内に残されていたとしても，必要のつど再呼び込みをする必要がある．

内容監視プログラムは，あるルーチンが主記憶装置に存在しているかどうかを判定するのに，**内容登録簿**（contents directory）を活用する．プログラム・ルーチンが主記憶装置に呼び込まれるつど，そのルーチンのための**記入項目**（entry）が登録簿の中に入れられる．したがって，内容登録簿は何時でも最新の状況を反映しているのである．また，LOADマクロ命令で呼び込まれたルーチンに関しては，記入項目が内容登録簿におさめられるだけでなく，**ロード・リスト**（load list）にも入れられる．

内容登録簿は，連係パック域と動的記憶域に存在するルーチンを表示する待行列の集まりである．つまり，連係パック域に対応する内容登録簿の待行列と各領域ごとの内容登録簿の待行列の集合体であり，システム待合せ域に収容されている．連係パック域用の内容登録簿の待行列に対するポインターはコミュニケーション・ベクトル・テーブル（CVT）に入れられているので，CVTをみることによって，その待行列をつきとめることができる．一方，領域用の内容登録簿の待行列をポイントしているのは，その領域を使用しているジョブ・ステップ・タスクのTCBである．

内容登録簿には，領域または連係パック域に収容されている各プログラムの**内容登録簿記入項目**(Contents Directory Entry; 略して**CDE**)が入っている．再入可能ないしは逐次再使用可能ルーチンの実行が終っても，そのルーチンが領域内に残っているかぎり，そのCDEは登録簿の待行列に収容されたままである．ところが，再使用不可能ルーチンの場合だと，そのオペレーションが完了すると，待行列から該当CDEを削除する仕組みになっている．それゆえに，前者の場合には，必要なルーチンがその領域に存在するかぎりくり返して利用されるが，後者の場合はそのつど必要なルーチンの呼び込みが行なわれるのである．

ロード・リストは，連係パック域ないしは該当領域にあるルーチンで，しかもLOAD マクロ命令を使って呼び出されたルーチンに対応する記入項目から構成されたリストである．それぞれの記入項目は，ロードされたルーチンに対応しており，かつそのルーチンのCDEをポイントしている．さらに，該当タスクの中でそのルーチンに対してLOAD マクロ命令が出された回数が記録されるようになっている．この回数は，当該ルーチンに対してDELETE マクロ命令が出されるつど1ずつ減算される．したがって，回数欄をみればその時点でこのルーチンを必要としている個数が判るのである．これによって，必要なルーチンを再三再四ローディングするという時間的手間をはぶいて，同一ルーチンを繰り返し利用できるように保障するのである．

以上のように，内容登録簿とロード・リストを用いて動的記憶域と連係パック域にある各種プログラム・ルーチンの状況を記録・保持し，必要に応じて非常駐ルーチンを主記憶装置に呼び込むのが内容監視のプログラム機能である．

6.　タイマー監視の機能

タイマー監視プログラム（timer supervisor）は，タイマーによる割込みと処理プログラムから要求される各種タイミング・サービスを行なう．オペレーティング・システムでは，日時を得るとか，時間間隔を計るとか，処理時間の制限を設けるとかの機能が提供されている．プログラマーがこれらの機能を要求するためには，TIME, TIMER, STIMER などのマクロ命令を指定すればよい．これらのマクロ命令が展開されて機械語になると，その中にはSVC 命令が含まれている．したがって，SVC 割込みが生じて，CPU の制御権はSVC 込み処理ルーチンに移り，さらにタイマー監視用のSVC ルーチンにブランチし，そこで適切な処理を行ってくれる．

一方，**間隔計時機構**（interval timer）の値が正から負に変ると（これが，一定の時間間隔が経過したことを意味するのであるが），タイマー／外部信号割込みが生じる．この割込みがタイマー割込みであることが判明すると，**タイマー／外部信号割込み処理ルーチン**（timer／external interruption handler）は制御権をタイマー監視ルーチンに渡す．このタイマー監視ルーチンは，**第二次**

タイマー割込み処理ルーチン（timer second-level interruption handler）と呼ぶことがあるが，このルーチンでは，特定時間間隔が経過したときに行なうように指示されている処理をしたあとで，間隔計時機構に新しい時間をセットする．

プログラマーが特定の時間間隔を指示しなくても，タイマー監視ルーチンは自動的に6時間の時間間隔をセットするので，6時間が経過するごとにタイマー／外部信号割込みが生じる．つまり，IPL後6時間経過すれば最初の割込みが生じる仕組みになっているのである．これは，タイマー監視ルーチンが**疑似時計**（pseudo-clock）と呼ばれる3個の主記憶装置の場所のうちの2個を更新するのに使う．また，タイマー監視ルーチンは深夜の12時になるとタイマー割込みが生じるようにしている．この深夜の割込みが生じると，**CVT**（Communication Vector Table）の中にある日付けを更新してくれるので，翌日の正しい日時が使えるようになる．

ここで，疑似時計についてふれておこう．疑似時計には3種類あり，いづれも主記憶装置内の特定ワードを用いている．疑似時計の一つは，**ローカル時間疑似時計**(Local Time Pseudo-Clock；略して**LTPC**)と呼ばれ，IPL時にSET指令で指示した時間の管理に使われる．2番目の疑似時計は，**24時間疑似時計**（24-Hours Pseudo Clock；略して**T4PC**）と呼ばれ，3番目のそれは，**6時間疑似時計**（6-Hours Pseudo Clock; 略して**SHPC**）である．T4PCは，6時間ごとの割込みが生じるつど6が加算される．ただし，T4PCがすでに18の値になっている場合には，6時間割込みが生じるとT4PCは自動的にゼロにもどされる．一方，SHPCには，経過時間が管理されており，6時間たつと自動的に割込みを生こさせるようにしている．したがって，SHPCに示されている時間は6時間を起えることはない．

上述のような周期的なタイマー割込みに加えて，プログラムで要求した時間間隔にもとづいて生じるタイマー割込みがある．この場合には，タイマー待行列を使って，時間間隔の長さと時間切れが生じる順番とを記録，更新するようにしている．

タイマー待行列は，システム待合せ域に作られ，一連の**タイマー待行列要素**（timer queue element）から構成される．各々の要素は，特定の時間間隔と関

連づけられており，時間間隔の長さと，時間切れになったときに取るべき措置とが記録される．したがって，時間間隔をセットすべき要求が出されるつど，1個の要素が作られタイマー待行列に入れられる．各要素は，時間切れが生じる順番に待行列に収容されるので，タイマー割込みが生じたときは，その待行列の最初の要素を見れば必要な措置が何であるかつかめる．ともあれ，割込みに対応する必要な処理が終れば，該当要素は待行列から取除かれるので，つぎに計測すべき時間間隔はタイマー待行列の最初の要素から入手できるのである．ところで，特定タスクと結びついたタイマー待行列の要素は，対応するTCB を双方からポイントするようになっている．このような要素はそのタスクが CPU の制御権を失うつど，いったんタイマー待行列から取除かれるようになっている．そして，そのタスクが再び制御権を得たとき待行列に該当要素を戻して，計測を再開するような方法が採用されている．したがって，この種の時間間隔は，完全にタスクが CPU を占有した正味の時間をもとにコントロールされるのである．

以上のように，処理プログラムから要求されるタイミング・サービスと，周期的割込みによる時間間隔処理およびプログラムで要求される時間間隔による割込み処理の機能を果すのが，タイマー監視プログラムである．

三. 割込み監視プログラムの機能構造

監視プログラムのオペレーションの全ては，割込みを契起として始まる．この割込みは，機械的な仕組みでありハードウェア的にひき起される．割込みの仕組みの詳細に関しては，第3章「オペレーティング・システムのためのハードウェア構造」の「割込みの概念」の項を参照して欲しい．ここでは，割込みが生じたことによって，割込み監視プログラムがどのような機能を果すかを中心に説明してゆく．

割込みが生じると監視プログラムが CPU の制御権を得て，割込みを起こしたタスクの遂行に必要な資源やサービスを提供する．なお，割込みには，実行中のプログラムが特別の要求を出すことによって生じるような計画的ないしは

意識的な割込みと，何らかの事象によって生じる無計画ないしは無意識的割込みとがある．いずれにしろ，割込み処理ルーチンでは，割込み時に引き渡された制御情報にもとづいて，その割込みを分析する．監視プログラムが必要とする制御情報のかなりのものは旧 PSW の中に集約されているので，これを使って割込みの分析をするのである．一方，新 PSW にはそれにふさわしい割込み処理ルーチンのアドレスが収容されているので，割込みが生じると自動的に適切な割込み処理ルーチンが実行され，上記の分析が行われるのである．つまり，割込みが発生すると，CPU は現 PSW を割込みのタイプに応じてあらかじめ定められた記憶域，すなわち旧 PSW に貯えたあと，新 PSW を CPU にロードする．新 PSW がロードされると，CPU は監視プログラム状態に切り換わり，CPU は新 PSW で指示されている割込み処理ルーチンに制御権を移すのである．割込み処理ルーチンは，割込みの原因を分析したあと，その割込みが要求しているサービスを遂行するにふさわしい監視プログラム部分に制御権を渡す．

すでに図 8-8 でも図示したように監視プログラムは，つぎの 5 種類の割込みの処理を行う：

① **SVC 割込み：** 監視プログラムは，どのような種類のサービスを要求しているのか判定し，それにふさわしいサービス・ルーチンに制御権を移す．
② **タイマー／外部信号割込み：** 監視プログラムは，割込み原因を究明し，タイマー・サービス・ルーチンか外部信号サービス・ルーチンのいずれかにブランチする．
③ **入出力割込み：** 監視プログラムは入出力監視プログラムにブランチし，そこで入出力エラーの処理ないしは入出力オペレーションの面倒をみる．
④ **プログラム割込み：** 監視プログラムは，割込みを起したタスクを異常終止させるか，ユーザー作成のエラー処理ルーチンにブランチする．
⑤ **機械チェック割込み：** 監視プログラムは，システムを待ち状態におくか，回復管理ルーチンにブランチする．

以上の 5 種類の割込みに関して，それぞれの割込み処理ルーチンがいかに機

能するかを詳述する．

1. SVC 割込み処理ルーチンの機能

システムないしユーザー・プログラムでマクロ命令を出すと，そのマクロ命令が機械語になった時点で，ほとんどの場合SVC命令を含んでいる．このSVC命令が実行されることによって，SVC 割込みが生じるのである．SVC 割込みは，いうまでもなくプログラムの中に意識的ないし計画的に組み込まれているものであり，監視プログラムに特定のサービスを要求するものである．SVC割込みによって直接制御権を受けとる監視プログラム部分が，**SVC 割込み処理ルーチン**（SVC interruption handler）である．

1） SVC 割込み処理ルーチンの主要な機能

SVC 割込み処理ルーチンは，主要な機能として下記に示すような事項を遂行する：

- 割込みを起こした呼び出しプログラムが用いていたレジスターの内容とSVC 用 旧PSW を保存する．
- **監視プログラム用要求ブロック**（Supervisor Request Block；略して**SVRB**）を作成すべきかどうかを決定する．所要の SVC ルーチンの実行中に別の割込みが生じたり，待ち状態が生じたとき，そのルーチンを再開始するためには SVRB が必要となるのである．
- 必要となれば，SVRB を作成し，該当ルーチンに関する情報を入れたうえで，現行タスク用の**要求ブロック待行列**（request block queue）の中に入れる．
- 所要の SVC ルーチンが主記憶装置に常駐しているかどうか判定する．
- 常駐 SVC ルーチンであることが判明すれば，それに制御権を渡す．
- 非常駐 SVC ルーチンであると，そのルーチンを補助記憶装置から呼び出し，制御権を受け渡す準備をする．
- 呼び出すことができないルーチンに対する要請を一時延期する．
- 呼び出しが可能になったとき，延期しておいた要請の実行を開始する．

上記の機能に加えて，SVC 割込み処理ルーチンは，つぎに示すような2つの機能も遂行する：

- SVC ルーチンが必要とする3つの制御ブロック，つまり CVT，呼び出しプログラムの TCB および現行 RB のアドレスを汎用レジスターに入れる．
- SVC ルーチンの実行が完了したときに制御権を返すアドレスを準備しておく．

SVC 割込み処理ルーチンは，**図 8-15** に示しているように機能的観点から2

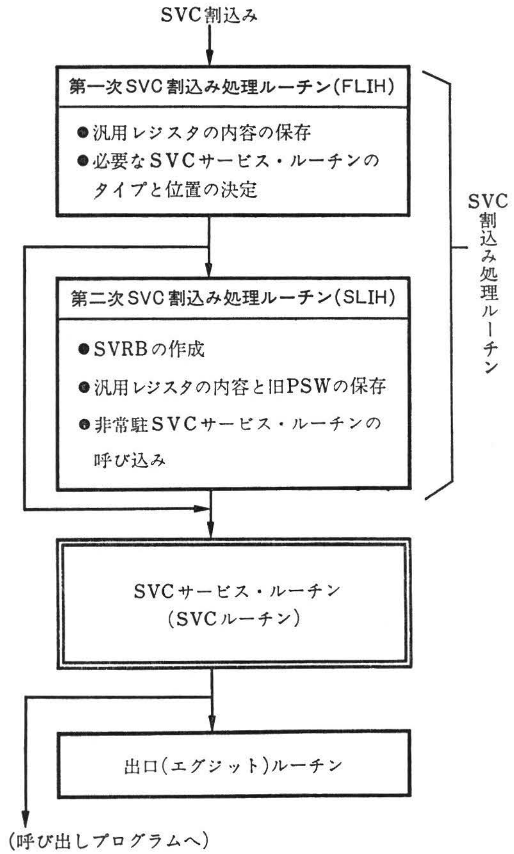


図 8-15 SVC 監視プログラムの制御権の流れ

つの部分に分けられる．つまり，**第一次SVC割込み処理ルーチン**(SVC First-Level Interruption Handler; 略して **SVC FLIH**)，**第二次SVC割込み処理ルーチン** (SVC Second-Level Interruption Handler; 略して**SVC SLIH**) である．SVC FLIHはレジスターの内容を保存し，所要のSVCルーチンのタイプと存在個所を判定する．ある種の使用頻度の高いSVCルーチンの場合には，SVC FLIHはSVCルーチンの実行を開始するために直接そのルーチンにブランチすることもある．このような性質以外のSVCルーチンに対しては，SVC SLIHの処理がさらに必要となる．SLIHは，SVRBを作成し，非常駐SVCルーチンの場合にはそのルーチンを呼び込んだうえで，制御権を受け渡す機能を果す．

以下で，SVC割込み処理ルーチンが遂行する主要な機能に関して補足説明をしておこう．

2）割込みルーチンの状況保存

SVC割込み処理ルーチンは，呼び出しルーチンに所属するレジスターの内容と旧PSWを保存しておかねばならない．というのは，割込み処理が終ったあとで，呼び出しプログラムに制御権を戻し，割込みを起した命令のつぎの命令から実行を再開しなければならないし，そのとき割込みが生じたときと同じレジスター内容が再現できないと，プログラムの実行に支障が生じるからだ．次に実行すべき命令のアドレスは現PSWにあり，これは割込みが生じたときにSVC用旧PSWに貯えられている．一方，レジスターの内容はSVC FLIHによって特別なレジスター保存域に貯えられる（**図8-16**参照)．

レジスターの内容および旧PSWは，実行されるSVCルーチンのタイプによっては，さらに別の保存域に移す必要性がある．もしも，SVCルーチンが直接的にも間接的にもSVC命令を出さないようになっていれば，再びSVC割込みは生じないので，SVC用旧PSWの内容も，特別なSVC用レジスター保存域の内容も破壊される心配はない．したがって，このようなSVCサービス・ルーチンの場合には，さらに別の保存域に移す必要は生じない．ところが，所要のSVCサービス・ルーチンの中で新たに別のSVC割込みを起すようになっていれば，SVC用旧PSWも，レジスター保存域も新たな割込みによっ

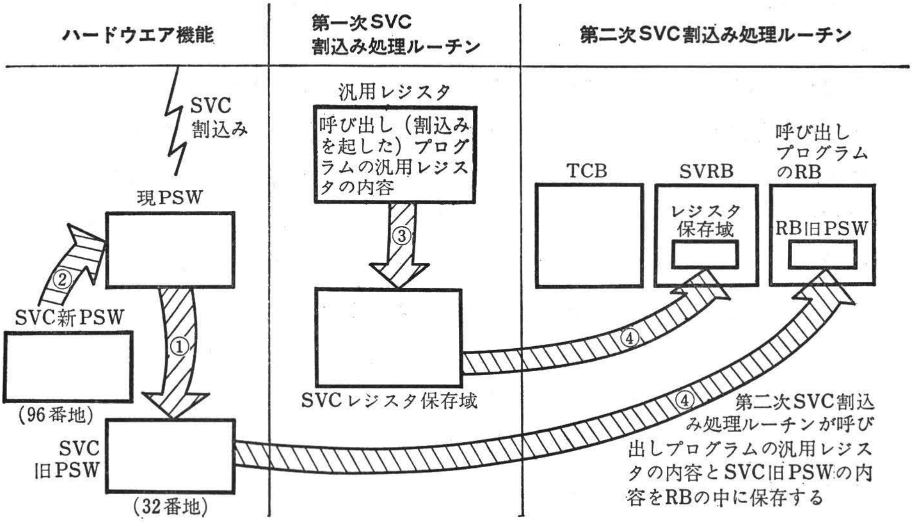


図 8-16 SVC 割込み処理ルーチンによる呼出しプログラムの状況保存の方法

て置きかえられてしまうので，別の保存域に移しておく必要がある．このようなオーバレイが起るのは，SVC ルーチンで SVC 命令を出したり，SVC 命令を用いたマクロ命令を出したり，あるいは割込みが生じて別のタスクに制御権が移って，そのタスクのルーチンで SVC 命令を出すといった場合である．いずれにしろ，このような状況が想定される SVC ルーチンを用いる場合には，図 8-16 で示しているように SLIH で別の保存域にレジスターの内容と旧 PSW を移すのである．

3） 監視プログラム用要求ブロックの必要性の判定

監視プログラム用要求ブロック（supervisor request block）を作成するかどうかの判断は，所要の SVC サービス・ルーチンが SVC 命令をだすことができる性格のルーチンであるかどうかにかかっている．一般に“タイプ I SVCルーチン”と呼ばれている共通的に使用される常駐 SVC ルーチンは，SVC 命令を用いることが許されていない．このタイプ以外の SVC ルーチンでは，SVC 命令を用いてもよいのである．つまり，SVRB を作成する必要があるか

どうかは，SVC ルーチンのタイプによって決まるのである．SVC ルーチンのタイプは，SVC テーブル・エントリのタイプ・ビットをテストすることで確認できる．SVC ルーチンがタイプⅠであると，SVC FLIH は，テーブル・エントリーに登録されているアドレスを用いてその SVC ルーチンに直接ブランチする．しかし，所要の SVC ルーチンがタイプⅠ以外だと，SVC FLIH は SLIH にブランチする．SVC SLIH は，監視プログラム用要求ブロック（SVRB）を作成して，そこに呼び出しプログラムのレジスターの内容や，SVC ルーチンを再開始するのに必要となる情報などを貯えるのである．

SVC SLIH では，所要の SVC ルーチンが主記憶域に常駐していて，ブランチ命令で直接に制御権が渡せるかどうか，そのルーチンが非常駐ルーチンであり，SVCLIB から呼び込む必要があるかどうかの判定をする必要がある．このために SVC SLIH は，SVC FLIH が受渡してくれた SVC テーブル・エントリーのタイプ・ビットをテストする．タイプⅠは先述した通りであるが，タイプⅡのルーチンは中核プログラムの中に常駐する性格がもたらされている．一方，タイプⅢとⅣは，IPL 時に**中核初期化プログラム**（nucleus initialization program）で主記憶域にあらかじめローディングする方法を取らないかぎり，非常駐ルーチンの性格が与えられており，必要のつど SVCLIB から呼び込む必要がある．なお，タイプⅢは，1個のロード・モジュールで構成されるルーチンであるのに対し，タイプⅣは，複数個のロード・モジョールからなっており，したがって，XCTL マクロ命令を使ってモジュール間の連係がなされている非常駐 SVC ルーチンである．

ところで，下記のような状態が生じたときには，SVC サービス・ルーチンの実行は一時中断されるか，その実行を延期せざるを得ない：

- SVC サービス・ルーチンの中で別の SVC 命令を出すことによって，SVC 割込みが生じたとき．
- SVC ルーチンが非常駐であり，かつ何らかの理由で直ちにそのルーチンをローディングすることができないとき．このときは，要求を一時延期することになる．
- SVC ルーチンを実行する前に，そのルーチンが入っている **SVC 一時域**

(SVC transient area) が別のルーチンでオーバレイされたとき.

- SVC ルーチンで即座に利用できないような資源を要求したとき. この場合には, その資源が利用可能になるまで待ち状態におかれる.

上記のいずれの場合にも, SVC サービス・ルーチンの実行に必要となる情報を一時貯えておく必要がある. つまり, 旧 PSW とか待ち回数などの情報が, SVC SLIH で作られた SVRB に貯えられるのである.

4) 非常駐ルーチンの呼び込み

SVC SLIH の中に, **一時域処理ルーチン** (Transient Area Handler ; 略して **TAH**) がある. このルーチンは, 中核プログラム域にある一時域を監視し, 必要に応じて補助記憶装置から非常駐 SVC ルーチンを呼び込む働きをする. 要求された SVC ルーチンが一時域に存在しないと, 一時域処理ルーチンは SVCLIB からそのルーチンを一時域に呼び込み, 御制権を渡すのである. しかし, 一時域のスペースが空いていないときは, そのスペースが空くまで, 該当 SVRB を待ち状態におく.

中核プログラム域には最低2個の一時域が設けられており, それぞれ1個の SVC ルーチンが収容できる. なお, 一時域の個数はオペレーティング・システムの生成時に指示することができ, 2個以上にすることが許されている.

非常駐 SVC ルーチンのいくつかを連係パック域に常駐させることも許される. つまり, 使用頻度が比軽的高い非常駐ルーチンを連係パック域に常駐させる方法をとれば, それだけ SVC ルーチンの呼び込み回数が少なくなり, その結果, 処理効率の向上が期待できるのである. このように要求された SVC ルーチンが連係パック域に存在する場合には, 一時域ルーチンは, 連係パック域用の内容登録簿の中の記入項目を探索し, そのルーチン名を見つけだす. そのあとで内容登録簿記入項目の中からエントリ・ポイントのアドレスを取り出し, そのアドレスを用いて該当ルーチンにブランチする.

以上のような機能を果すのが, **SVC 割込み処理ルーチン** (SVC interruption handler) である. これをまとめると, **図 8-17** のようになる.

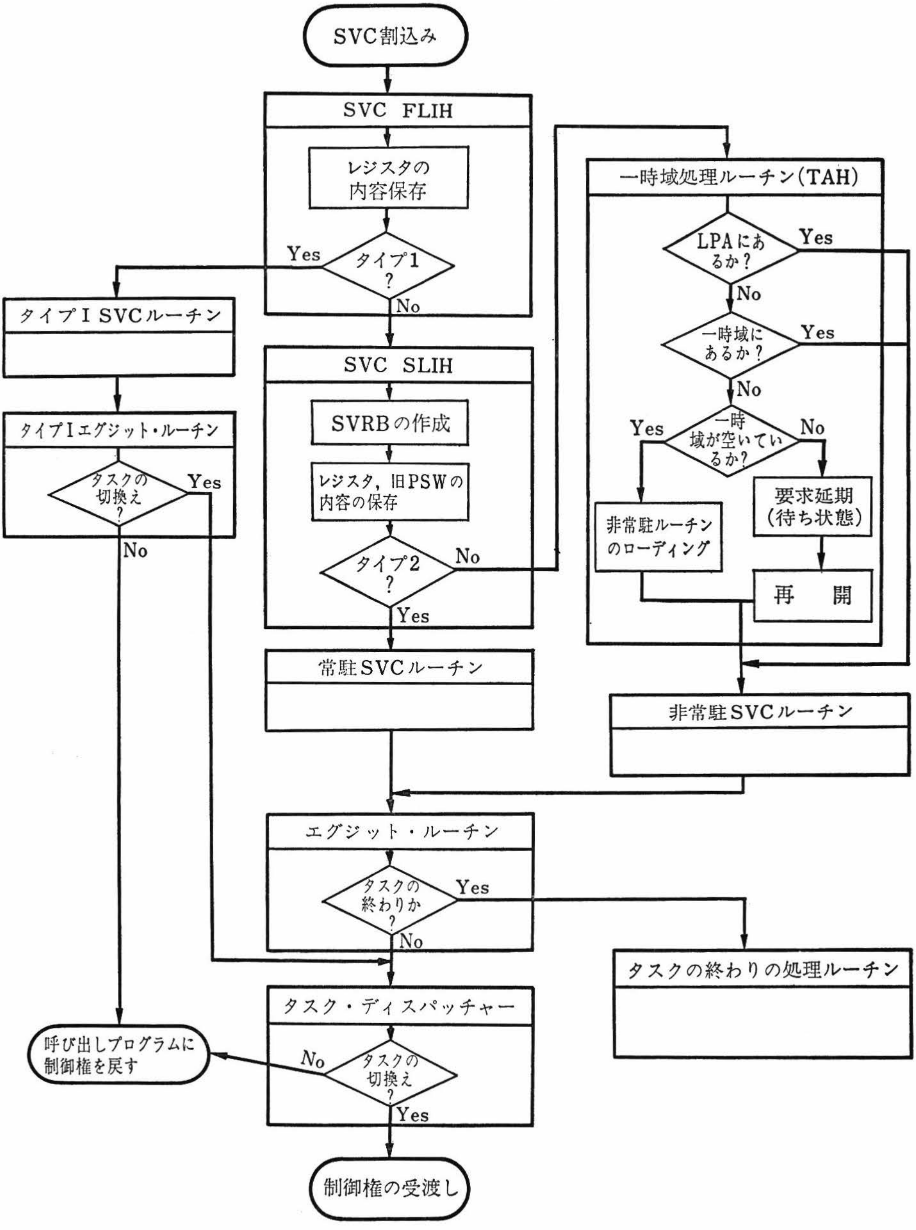


図 8-17　SVC 監視プログラムの機能構造

2. プログラム割込み処理ルーチンの機能

指定違反の命令や不適切な命令の実行が行われると，プログラム割込みが生じ，その原因コードはプログラム割込みの旧PSWにおさめられる．プログラム割込みの原因には，不適当なアドレス指定，許されない命令コードの使用，特権命令の実行，不適切なデータの使用，演算結果のオーバフローやアンダフロー，あるいは記憶保護違反などがある．

図 8-18 で示しているように，プログラム割込みが生じると，CPU の制御権は自動的に**プログラム割込み処理ルーチン** (program interruption handler) に受け渡される．このルーチンでは，割込みを起したのが監視プログラムかそれとも処理プログラムなのかを旧PSWの状態コードをテストすることによって確かめる．監視プログラムの中でプログラム割込みが生じたのであれば，それ

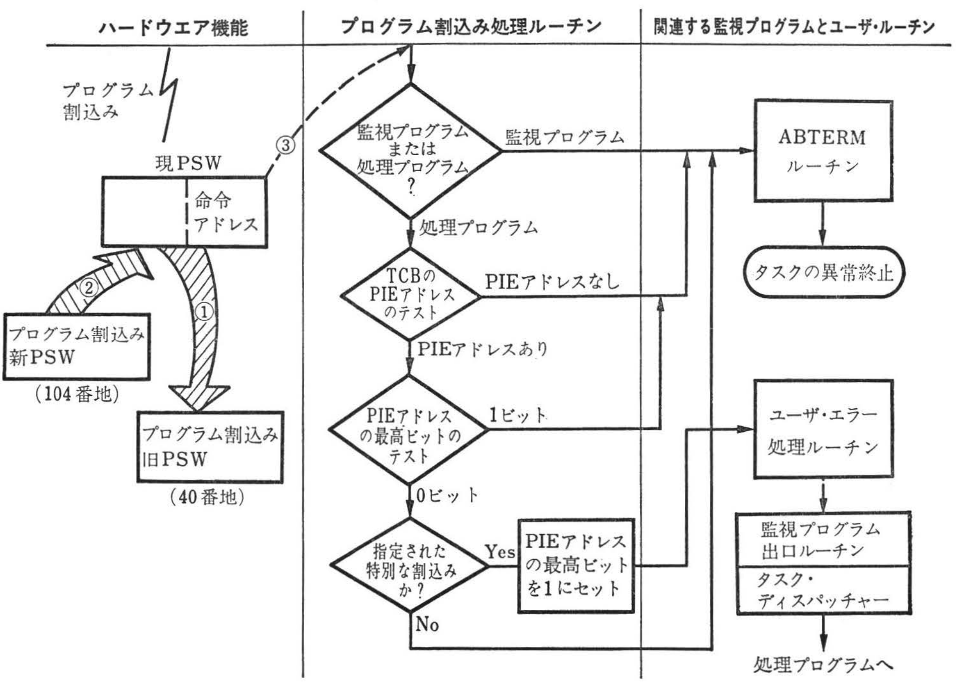


図 8-18 プログラム割込みの処理構造

以上のプログラムの実行は望ましくないので，制御権を監視プログラムの中のABTERM ルーチンに渡す．ABTERM ルーチンでは，割込み時に実行していたタスクを強制的に終了させてしまう．

一方，処理プログラムの中でプログラム割込みが生じたのであれば，割込み処理ルーチンは，TCB を調べて**プログラム割込み要素** (Program Interruption Element; 略して **PIE**) のアドレスが挿入されているかどうかを確認する．というのは，PIE がプログラム割込みの処理を行うユーザー・ルーチンに関係した制御ブロックであるからだ．ユーザーはプログラム割込みに関して独自の割込み処理ルーチンを作成し，処理プログラムに組み込むことが許されている．このために，SPIE マクロ命令を用いて PIE の内容設定をするのである．SPIE マクロ命令では，割込み処理を行うエラー処理ルーチンのアドレスと，どの種のプログラム割込みに関してそのルーチンを適用するかを指示する．この SPIE マクロ命令が実行されることによって，PIE が作りあげられると同時に PIE のアドレスが TCB の所定の位置に挿入されるのである．TCB にこのアドレスが挿入されているということが，ユーザー処理ルーチンを適用することを暗示しているのである．

TCB に PIE のアドレスが挿入されていないときは，ユーザー・エラー処理ルーチンが存在しないことを意味するので，監視プログラム内でプログラム割込みが生じた場合と全く同じ取扱いをする．つまり，ABTERM ルーチンに制御権を移すのである．一方，PIE アドレスがみつかったとなると，さらにそのアドレスを収容しているワードの最高位のビットが1にセットされているかどうかテストする．というのは，制御権がユーザー・エラー処理ルーチンに渡るときにはいつでもこのビットを1にセットすることによって，ユーザーのエラー処理ルーチンで2番目のプログラム割込みが生じた場合には，そのタスクを終了させてしまうように目論んでいるのである．したがって，このビットをテストし1であることが判明すると，ABTERM ルーチンに制御権を渡すのである．

TCB に PIE アドレスがあり，さらに最高位のビットが0にセットされているなら，プログラム割込み処理ルーチンは，目下のプログラム割込みの原因が，SPIE マクロ命令の中で指示されている種類の割込みであるかどうかを確

める. SPIE マクロ命令で指定された割込みでないとすると, ユーザーのエラー処理ルーチンで面倒みきれない種類の割込みであるので, ABTERM ルーチンに制御権を渡し, そのタスクを終了させてしまう. 逆に, 指定された特別な割込みに該当する場合は, はじめてユーザー・エラー処理ルーチンに制御権を受け渡す. 先述したように制御権を渡すまえに TCB の PIE アドレスの最高ビットを1にセットする. なお, ユーザー・エラー・ルーチンの処理が完了すると, 監視プログラムの出口ルーチンを経由したうえで, タスク・ディスパッチャーが割込みを起したプログラムに制御権を戻してくれるのである.

3. 外部割込み処理ルーチンの機能

タイマー／外部信号割込みが生じると, 現 PSW は外部割込み旧 PSW に貯えられ, その代りに外部割込み新 PSW が現 PSW になる. その結果, CPU の制御権は**第一次外部割込み処理ルーチン** (external first-level interruption handler) に移される.

外部 FLIH は, まず汎用レジスターの内容を割込みを起したタスクの TCB の中に保存する. **システム管理機能** (System Management Facility ; 略して **SMF**) が組み込まれている場合には, このあと制御権を SMF の**待ち時間収集ルーチン** (wait time collection routine) に移す. 割込みを起したタスク, つまり現行タスクの TCB が待ち状態を示していれば, このルーチンでシステムの待ち時間を収集する. 一方, 現行タスクの TCB が活動状態なら, 待ち時間収集ルーチンは何の機能も果さない. いずれにしろ, このルーチンから制御権は再び外部 FLIH に戻される.

外部 FLIH は, 外部割込み旧 PSW を現行 RB に保存したうえで, 旧 PSW を調べる. これによって, 外部割込みを起した原因をつきとめる. タイマーによる割込みであると, **タイマー第二次レベル割込み処理ルーチン** (timer second-level interruption handler) に制御権が移され, 時間間隔に関する必要な処理準備が行なわれる. 一方, オペレータがコンソール上の割込みキーを押したことによって生じた割込みであると, **常駐外部信号処理ルーチン** (resident external interruption handler) に移り, コミュニケーション・タスクを遂行す

るための準備を行う．

それぞれの割込み処理が終ると，再び外部FLIHに制御権が戻される．システムが一時的に割込み禁止の状態にされている場合だと，FLIH は外部割込みの旧PSWを使って制御権を直接に割込みを起したプログラムに戻してしまう．ところが，システムが割込み状態にされていないときには，タスク・ディスパッチャに制御権を渡す仕組みになっている．

以上のような外部割込みの処理は，**単一 CPU システム**（uni-processing system）の場合である．**双頭システム**（multi-processing system）の外部割込み処理は，双方の CPU からの信号によってのみ外部割込みの処理が付加されるので，単一 CPU システムの場合とは大巾に処理構造が異ってくるのであるが，ここでは特にふれないことにする．

4. 入出力割込み処理ルーチンの機能

入出力割込み処理ルーチン（I/O interruption handler）の主要な機能は，入出力監視プログラムへの橋渡しを行うことである（**図8-19**参照）．すべての入出力サービスや入出力エラーの処理は，入出力監視プログラムの中で行われるからである．なお，入出力割込み処理および入出力プログラムに関する詳細については，第3章「オペレーティング・システムのためのハードウェア構造」の「入出力オペレーションと監視プログラムの働き」の項目，および第5章「データ管理の諸機能とその構造」の「データ管理機能のまとめ」の項目を参照されたい．

5. 機械チェック割込みと回復管理機能

プログラミング資源およびハードウェア資源を最も効率よく使用するためには，ハードウェアの誤動作によって引き起こされる予期しない割込みの時間および影響を最少限にくいとめる必要がある．コンピュータ・システムの効率を高めるためには，ハードウェアの誤動作が発生した時点のシステムの状況を刻明に記録すると同時に，そのデータを分析して可能なら自力でその誤動作を訂正し，オペレーションを続行することが望まれる．オペレーティング・システ

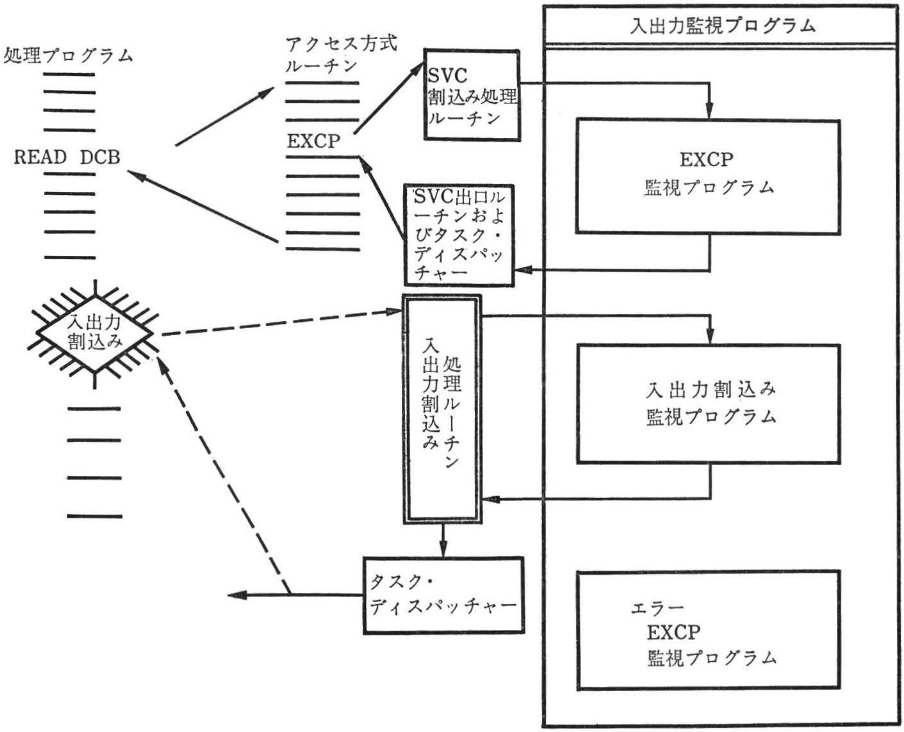


図 8-19 入出力割込み処理ルーチン

ムには，このような機械の誤動作に対処するため，**回復管理機能**（recovery management facility）が用意されており，ハードウェアの誤動作が発生した時点のシステム環境に関するデータをとったり，このデータを分析して妥当な回復操作を行って，システム効率を高めることに貢献するよう配慮されている．この場合，機械の誤動作を修正し，回復させるレベルには，エラーの大きさと範囲によってつぎの4つが考えられる：

① **ハードウェア機能の回復と処理の続行：** 割込みを起した命令または入出力操作を再試行し，正しく作動することが確認されることによって処理をそのまま続行する．

② **システムの回復と後続ジョブの続行：** 誤動作の回復が不可能であっても，それが他のタスクの処理に影響を及ぼさないものと診断される場合に

は，その影響をうけるタスクを捜し出して，関係するジョブだけを中止し，システムのオペレーションは続行する．

③ **システムの再始動：** IPL 手順によるシステム・オペレーションの再始動．

④ **システムの修復：** 回復不可能な機械の故障を，回復管理機能で記録した情報を活用して究明し，短時間で診断，修理を行なう．

一般的には上記のようなレベルがあるが，機械チェック割込みが生じたときの処理は，どの回復管理機能を適用するかによってかなりの差がある．オペレーティング・システムに用意されている回復管理プログラムには，つぎに示すように，3種類の機械チェック回復プログラム，1種類のチャネル・エラー回復プログラム，2種類の入出力回復プログラムがある：

- システム環境記録ルーチン0 (System Environment Recording 0 routine; 略して **SER 0**)
- システム環境記録ルーチン1 (System Environment Recording 1 routine; 略して **SER 1**)
- 機械チェック処理ルーチン (Machine Check Handler ; 略して **MCH**)
- チャネル・チェック処理ルーチン (Channel Check Handler; 略して **CCH**)
- 代替経路再試行ルーチン (Alternate Path Retry ; 略して **APR**)
- 動的装置再構成ルーチン(Dynamic Device Reconfiguration;略して **DDR**)

これら回復管理機能に関して，機械チェックと入出力チェックに大別して，以下で説明することにする．

1) 機械チェックの回復管理機能

機械の誤動作によって予期しない割込みが生じると，直ちに CPU の制御権は回復管理ルーチンに渡される．

つぎのような回復管理ルーチンのうちのいずれか（つまり，システムに組み込まれているルーチン）に制御権を移し適切な措置を委ねるのである：

- 回復管理ルーチンが組み込まれていないとき：　機械チェック割込み用

の新 PSW がロードされ，CPU は待ち状態に入る．

したがって，オペレータは，**システム環境記録・編集・印刷プログラム**(System Environment Recording, Editing, Printing Program ; 略して**SEREP**) をローティングし，CPU のログアウト域の内容を印刷し，それをもとに原因を分析する．なお，SEREP はオペレーティング・システムとは無関係に実行できる独立した診断用プログラムである．

- SER 0 の回復管理ルーチンが組み込まれているとき： 機械チェック割込み用の新 PSW がロードされ，CPU の制御権が SER 0 ルーチンの常駐部分に引き渡される．このルーチンではその割込みに関係するシステム環境データを記録する．この仕事が完了した時点で，CPU を待ち状態におく．
- SER 1 の回復管理ルーチンが組み込まれているとき： 機械チェック割込み用の新 PSW がロードされ，CPU の制御権は SER 1 ルーチンに移される．このルーチンでは，システム環境データを記録したうえで，割込み原因をチェックし，その機械チェックが影響するジョブ・ステップだけを異常終止させてその後の処理を続行するか，CPU を待ち状態におくかのいずれかの手順を取る．
- MCH の回復管理ルーチンが組み込まれているとき： 機械チェック割込み用の新PSWがロードされ，制御権は**機械チェック処理ルーチン**(machine check handler) に移される．このルーチンは，システム環境データの記録をとったあと，エラーを分析し，処理が行われなかった命令の再試行を行ない，回復を試みる．また，再試行が無理だとか，再試行が失敗に終ったときは，誤動作の修復を試みる．

CPU または主記憶装置にエラーがあると，MCH は制御権を受け取り，制御プログラムに関連するエラーか，それとも処理プログラムに関連するエラーかを判定し，それによって異った回復手順をとる．いずれにしろ，システム環境データをとり，このデータを分析することによってどのレベルで回復を行なうことができるかを決定する．

MCH の回復管理機能は，かなり複雑なものであるが，これを機能的に

整理して図式化すると**図 8-20** に示したような回復手順になる．

2)　入出力エラーの回復管理機能

入出力エラーは，機械チェック割込みを引きおこすのではなく，入出力割込みを引き起す．しかし，入出力監視プログラムは，これらエラーの割込みが生じると，機械チェック割込みの場合と同様に回復管理ルーチンに制御権を移し，その処理を委ねる．したがって，便宜上機械チェック割込みの処理として説明する．チャネルあるいは入出力装置に関するエラーの場合の処理も，機械チェックの場合と同じように，適用される回復管理機能の種類によって，かなりの差がある．以下でこの場合の回復管理機能について説明する：

- 回復管理ルーチンが組み込まれていないとき：　入出力監視プログラムが，機械チェック割込み用新 PSW をロードし，CPU を待ち状態におく．
- SERO, SER 1 ないしは MCH が組み込まれているとき：　入出力監視プログラムが，機械チェック割込み用新 PSW をロードし，制御権を回復管理ルーチンに渡す．回復管理ルーチンは環境データの記録をとったあとで，CPU を待ち状態におく．
- **チャネル・チェック処理ルーチン** (channel check handler) が組み込まれているとき：　チャネル・エラーが生じたとき，入出力監視プログラムは CCH に制御権を渡し，そこでチャネル・エラーの状態をできるだけ修復するように試みる．このCCHには2つの主要な機能が持たされている．その1つは，チャネルに関するログアウト・データを分析・整理して一定の形式にまとめ，**エラー回復手順プログラム** (Error Recovery Procedure; 略して **ERP**) に提供することである．このデータをもとに，それにふさわしい ERP が取り出され，エラーの生じたオペレーションを再度試みるための諸準備を行なうのである．オペレーションの再試行自体は ERP が行うのではなく，当然入出力監視プログラムが面倒みるのであるが，そのためのお膳立てを ERP が行うのである．もう1つの主要な機能は，チャネル・エラー時のシステムの様子，つまりシステム環境データを**チャネル・エラー内部記録エントリー** (channel error inboard record entry) と呼ば

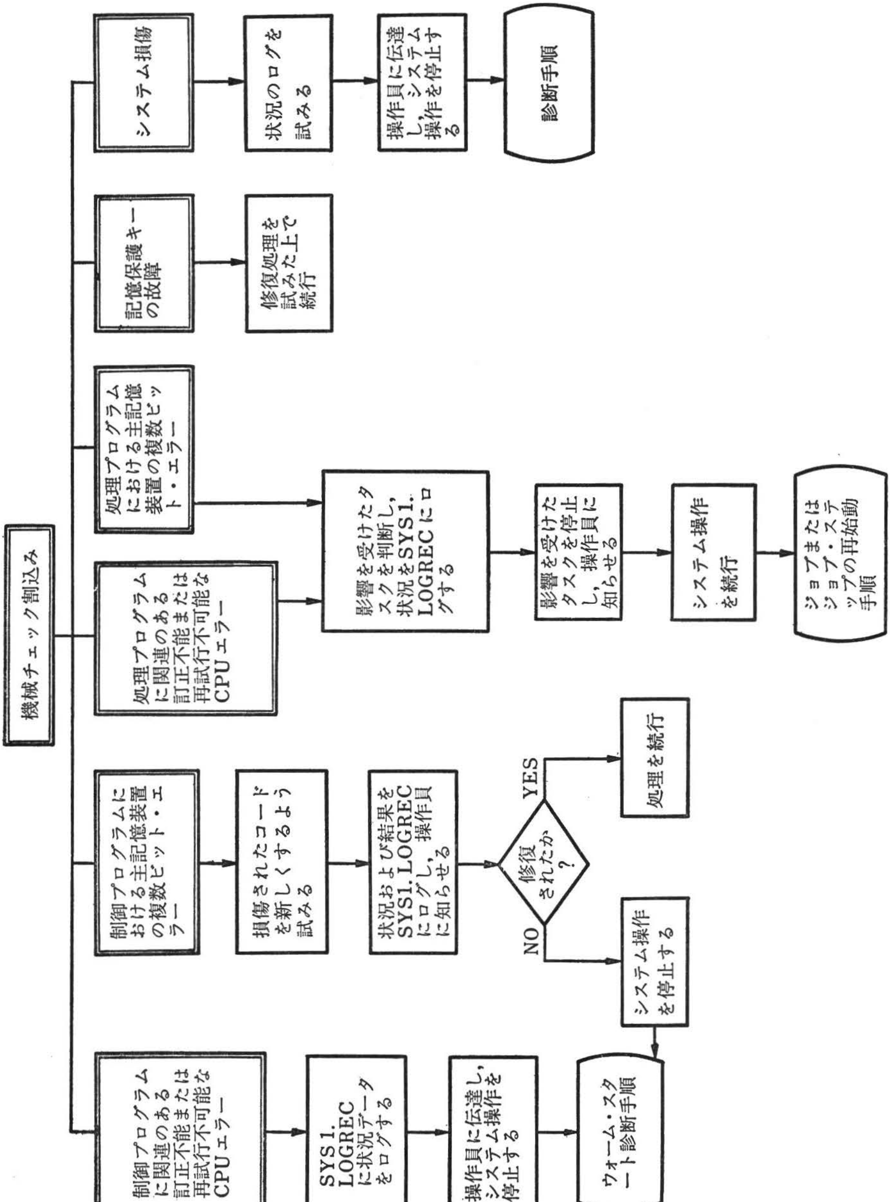


図 8-20　MCH による機械チェックの回復手順の流れ

れる場所に記録することである．この記録エントリーはあとで入出力監視プログラムの中の**外部記録ルーチン**（outboard recorder routine；略して**OBR**）がSYS 1. LOGREC上に書き出してくれる．なお，入出力オペレーションに関する統計的なデータを書き出す機能を果たす入出力監視プログラム部分は，特に**統計データ記録ルーチン**（Statistic Data Recorder Routine；略して**SDR**）と呼び，OBRと区別している．

以上の回復管理ルーチンに加えて，**動的装置再構成ルーチン**（DDR）と**代替経路再試行ルーチン**（APR）の機能がある．

永続的な入出力装置のエラーが生じたとき，DDRがシステムに組み込まれていると，OBR/SDRは制御権をDDRに渡す．DDRは，永続的な入出力エラーを起した装置にあるボリュームを別の装置に乗せかえるように操作員に指示し，そのボリュームの切り替えが行なわれると，切り替えられた装置を使って入出力オペレーションの再試行を試みるのである．

APRは，チャネル・エラーを間接的に回復するための手段である．すなわち，あるチャネルを使って生じた入出力オペレーションを別のチャネルに切り替えて再試行するための機能である．

以上みてきた入出力オペレーションに関する回復管理機能をまとめて，手順化して示すと，**図8-21**のようになる．

3)　システム環境記録に関する補足

システム環境記録ルーチン(System Environment Recording；略して**SER**)は，CPUおよび入出力チャネルの誤動作の記録をとるための制御プログラム・ルーチンである．すでに説明したように，SERにはSER 0とSER 1の2種類がある．どちらを選択するかは，オペレーティング・システムの生成時にきめられる．

機械チェック割込みが生じたときに，このSER機能が回復管理プログラムとして選択されていると，制御権はSERに移される．一方，チャネル・エラーが生じたときも，入出力監視プログラムが制御権をSERに渡す．

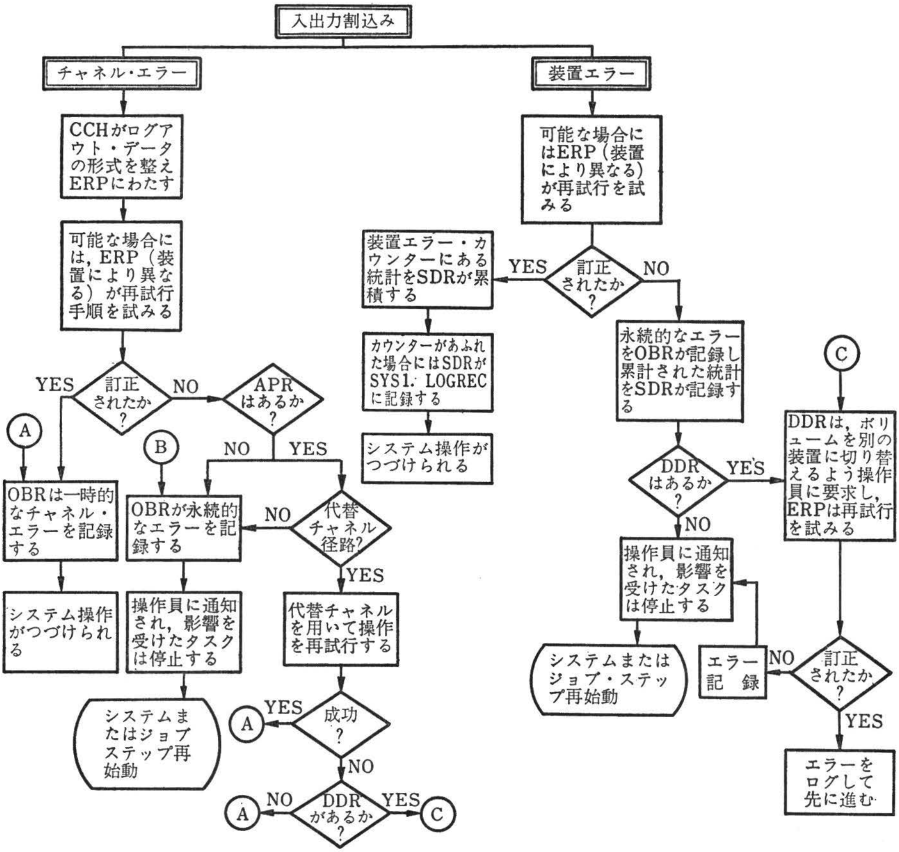


図 8-21　入出力割込み後のエラー回復手順の流れ

SER 0 の場合だと，誤動作のタイプを判断し，可能であるなら，SYS 1. LOGREC と呼ばれるシステム・データ・セットにそのエラーに関する状況を示したレコードを書き出す．このレコードの書き出しが完了すると，オペレータに対して制御プログラムの再ローディングを促すメッセージを印刷したあと，CPU を待ち状態にする．SER 0 でエラーの状況記録の書き出しができないと判断した場合には，オペレータに SEREP を利用するよう指示したメッセージを書き出して，待ち状態に入る．

SER 1 も，SER 0 と同じようにエラーが生じたさいのシステムの状況を示

すシステム環境データを収集して，SYS 1. LOGREC に書き出す．さらに，SER 1 は**選択的中止分析**（selective termination analysis）を行なう．これは，エラーと特定タスクを結びつけることを試みるものである．タスクとエラーの間の関係がみつかり，SER 1 の処理中に特別な誤動作が発生しない場合には，次の措置がとられる：

① エラーに関係したタスクの遂行を中止する．つまり，そのタスクだけを異常終止させる．
② エラーに関する環境データを一定形式に集約し，SYS 1. LOGREC に書き出す．
③ システムのオペレーションを続行する．

エラーと特定タスクとを結びつけることができない場合，制御プログラムが損傷を受けている場合，あるいは SER 1 の実行中にさらに別の誤動作が発生してエラー・レコードを完全に書き出すことができないような場合には，オペレータに制御プログラムの再ロードを要求するメッセージを出して，システムを待ち状態に入れる．このメッセージが出されたときは，システムを初期設定する必要があり，IPL からやり直すことになる．

なお，各種エラーに関する記録や入出力オペレーションに関する統計的データが出力されている SYS 1. LOGREC の内容を取り出したときは，**環境記録編集・印刷ルーチン**（Environment Recording Edit and Print routine; 略して **EREP**）を用いる．このルーチンは，SER で記録された SYS 1. LOGREC 上のレコードを一定形式に整えて，印刷装置ないしは磁気テープ装置に書き出してくれる．

四. タスク監視プログラムの機能構造

監視プログラムは，ジョブ制御プログラムの中の開始／終止プログラムの要求にもとづいて**ジョブ・ステップ・タスク**（job step task）を生成する．また，すでに存在するタスクにサブタスクを Attach するための要求に応じてサ

ブタスクを生成する．さらに，どのような順序でタスクを実行すべきかを決定するのも監視プログラムである．

システムが実行するタスクのそれぞれは，**タスク制御ブロック**（Task Control Block；略して **TCB**）で表示される．TCB には，該当タスクにかかわりをもつ制御情報とそのタクスの状態を表示する情報とが集約されている．**図 8-22** に示した TCB のうち斜線を入れた部分がこれらの情報にあたる．さらに，

バイト				
−32	TCBFRS：浮動小数点レジスターの保存域			
0	TCBRBP：要求ブロック(RB)のアドレス			
4	TCBPIE ：プログラム割込み要素(Program Interrupt Element：PIE)のアドレス			
8	TCBDEB：DEB待行列のアドレス			
12	TCBTIO ：タスク入出力テーブル(TIOT)のアドレス			
16	TCBCMP：標識とタスク完了コード			
20	TCBTRN：制御コア・テーブルの標識とアドレス			
24	TCBNROC：ロールアウト資格	TCBMSS ：最後のSPQEのアドレス		
28	TCBPKF ：記憶保護キー	TCBFLGS：タスク終了，指名可能性，その他の標識		
		TCBLMP：制限優先順位	TCBDSP：指名順位	
36	TCBLLS：LOADマクロ命令でロードされたプログラムのロード・リスト要素(LLE)のアドレス			
40	TCBJLB ：JOBLIB DCBのアドレス			
44	TCBJPQ：ジョブ・パック域(JPA)に対する内容登録簿エントリー(CDE)のアドレス			
48	TCBGRS：汎用レジスターの保存域			
112	TCBQEL：Enqueueの数	TCBFSA：最初のプログラム保存域のアドレス		
116	TCBTCB：次の低い優先順位のTCBアドレス			
120	TCBTME：タイマー待行列要素(Timer Queue Element：TQE)のアドレス			
124	TCBJSTCB：ジョブ・ステップに対する最初のTCBのアドレス			
128	TCBNTC：サブタスク用待行列上の先行TCBのアドレス			
132	TCBOTC：親タスクのTCBのアドレス			
136	TCBLTC：サブタスク用待行列上の最後のTCBのアドレス			
140	TCBIQE ：ETXRルーチンに対するIQE(Interruption Queue Element)のアドレス			
144	TCBECB：タスク完了が知らされるECBのアドレス			
148	TCBTSFLG：タイム シェアリングの標識	TCBSTPCT：STATUSマクロ命令の回数カウント	TCBTSLP：タイム シェアリング用タスクの制限優先順位	TCBTSDP：タイム シェアリング用タスクの指名順位
152	TCBPQE：領域用ダミーPQE(Partition Queue Element)から８引いたアドレス			
156	TCBAQE：割当て待行列要素(Allocated Queue Element：AQE)のアドレス			
160	TCBNSTAE：現STAE制御ブロックのSTAE標識			
164	TCBTCT：TCT(Timing Control Table)のアドレス			
168	TCBUSER：ユーザー用フィールド			
172	TCBDAR：DAR標識	TCBNDSP1：指名不能ビット	TCBNDSP2：指名不能ビット	TCBNDSP3：予約フィールド
176	予約フィールド			
180	予約フィールド	TCBJSCB：JSCBのアドレス		

図 8-22 タスク制御ブロック（TCB）の形式と内容

TCB にはタスクを実行するために割り振られたシステム資源を示すための各種のポインター（アドレス）が収容されている.

TCB は，タスク遂行にあたっての中枢神経に相当するものである．制御プログラムが何らかの機能を果すために必要となる情報の大半が，この TCB を通して得られる仕組みになっている．入出力監視プログラムが用いる DEB や TIOT などのアドレスが示されている．内容監視プログラムのための**ロード・リスト要素**（Load List Element ; 略して **LLE**）と**内容登録簿エントリー**（Contents Directory Entry ; 略して **CDE**）のポインターがある．主記憶装置監視プログラムが用いる**プログラム待行列要素**（Program Queue Element ; 略して **PQE**）や**サブプール待行列要素**（Subpool Queue Element; 略して **SPQE**）などのポインターもある．タイマー監視プログラムのためには，**タイマー待行列要素**（Timer Queue Element ; 略して **TQE**）のポインターがある．さらには，タスク監視プログラムが必要とする**要求ブロック**（Request Block ; 略して **RB**），**割込み待行列要素**（Interruption Queue Element; 略して **IQE**）などがある．これらのポインターの関係を図式化すると**図 8-23** のようになる.

オペレーティング・システムのキーとなる TCB は，システム生成時に作られる．これら TCB には，マスター・スケジューラー用タスク，システム・エラー用タスク，ロールアウト／ロールイン用タスク，コミュニケーション用タスク，さらには**一時域取出し用タスク**（transient area fetch task）がある．この種のキーになる TCB 以外のタスク制御ブロックのすべては，制御プログラムないしはユーザー・プログラムからの要請にもとづいて，監視プログラムの中の**タスク生成ルーチン**（attach routine）が構築してくれる．マスター・スケジューラー用タスクは，最大15個までの開始／終止プログラム用タスクを取り附けることができる．一方，開始／終止プログラム用タスクはジョブ・ステップ・タスクを取り附けるのである．このようにして制御プログラム用タスクと処理プログラム用タスクが形作られるのである（**図 8-24** 参照).

システム内のすべての TCB は，**指名順位**（dispatching priority）にもとづいて相互に連鎖され，TCB 待行列を構成する（**図 8-25** 参照)．一時域取出し用 TCB が最も指名順位の高い TCB である．そのあとシステム・エラー用

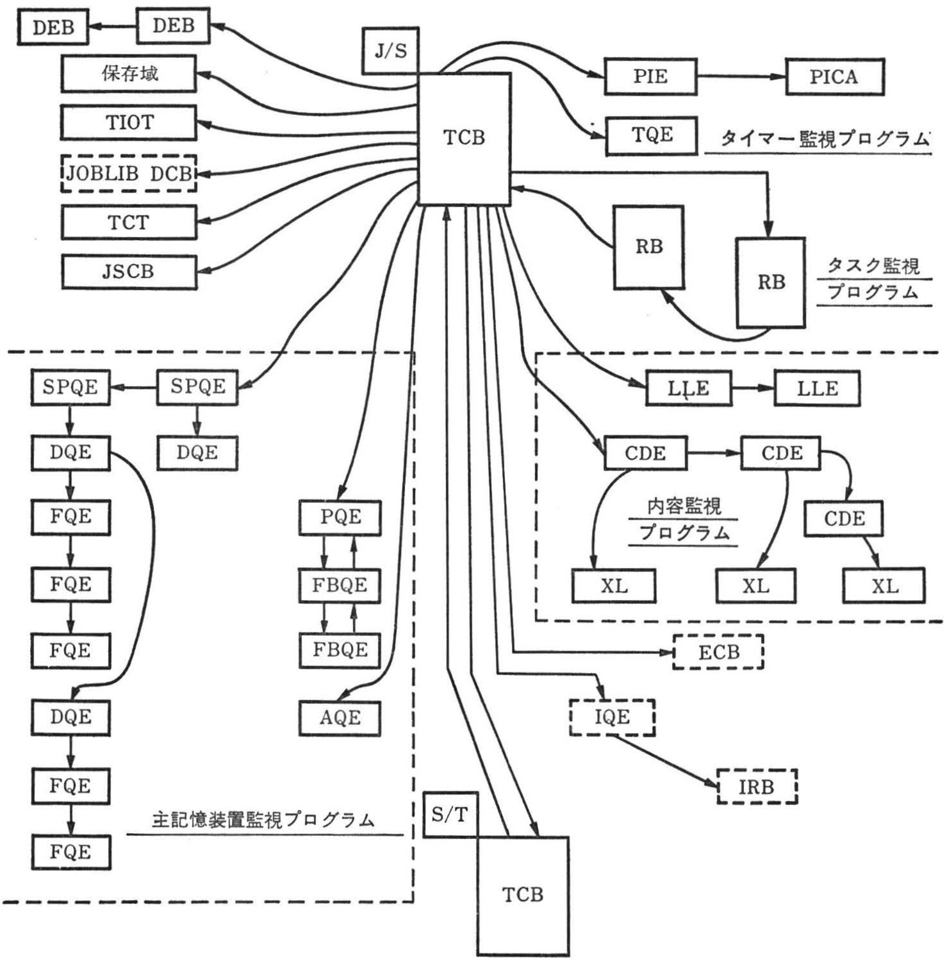


図 8-23 制御ブロックへのポインター

TCB，ロールアウト／ロールイン用 TCB，コミュニケーション用 TCB，そしてマスター・スケジューラー用 TCB と続く．なお，この他のタスクの指名順位は，後述するように ATTACH マクロ命令で指示したパラメータにもとづいて監視プログラムで割り当ててくれる．TCB 待行列に同じ優先順位の TCB が複数個ある場合には，**先入れ－先出し法（FIFO）**にしたがう．

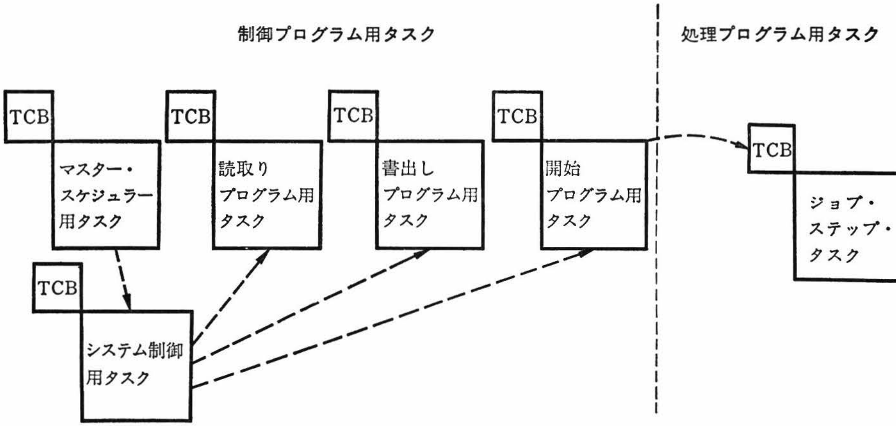


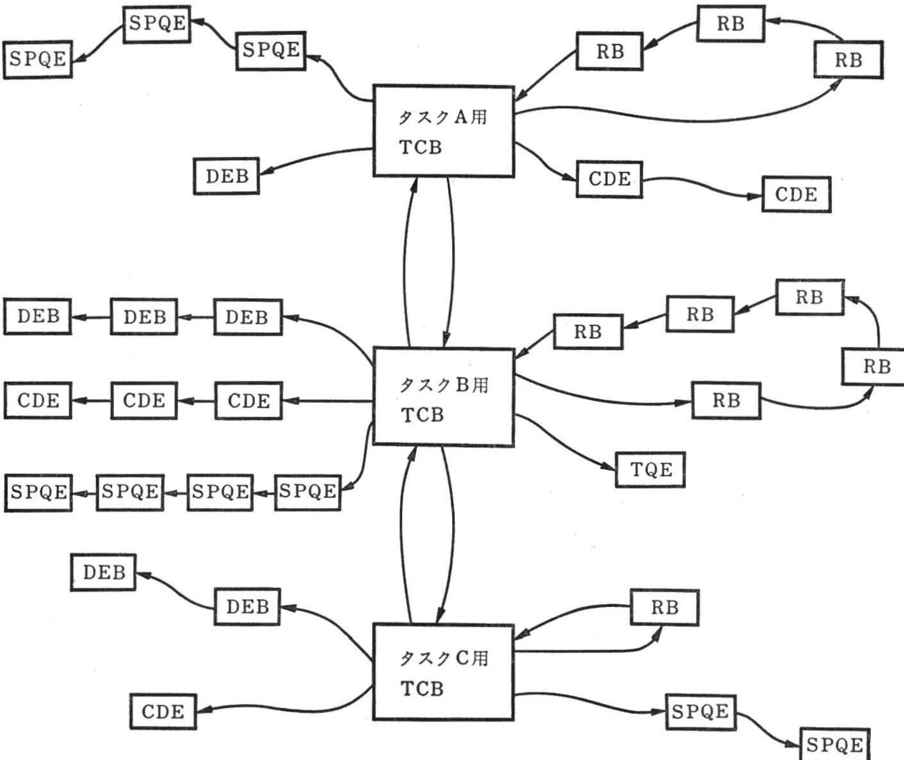
図 8-24　制御プログラム用タスクと処理プログラム用タスク

タスク監視プログラムは，タスクを実行する順序にしたがって制御してゆく．この制御を行なうのが，タスク監視プログラムの中の，**ディスパッチャー**（dispatcher）であり，TCB 待行列をもとにタスクの切換え制御を遂行する．ところで，TCB 待行列の中で最も優先順位の高いタスクが，最も実行順序の高いタスクともいえない．というのは，そのタスクが何かの事象を待ったり，逐次使用が要求されている資源を待っている場合があるからだ．前者は，WAIT マクロ命令を通して，後者は ENQ マクロ命令を通して指示されるのが普通である．ということから，TCB 待行列はシステム内の全タスクの状況記録の機能も果しているのである．

タスク監視プログラムの機能は，大別してつぎの3つのカテゴリーに区分できる．1）タスク制御ブロックに直接関連したサービス機能，2）タスク制御ブロックに間接的なかかわりをもったサービス機能，3）監視プログラムに対する内的サービスの3種類である．

TCB に関する直接サービスには，TCB の生成，処理ないしは除去の仕事が含まれる．これらサービスは，つぎの4つの要素からなる：

- サブタスクの**取り附け**（attach）
- タスクの指名順序の変更
- TCB の中の特別な情報の取出し



SPQE—主記憶装置監視プログラムが用いる待行列要素
RB　—タスク監視プログラムが用いる待行列要素
CDE　—内容監視プログラムが用いる待行列要素
DEB　—データ管理プログラムが用いる待行列要素
TQE　—タイマー監視プログラムが用いる待行列要素

図 8-25　TCB 待行列の基本構成

- サブタスクの**消去**（detach）

TCB に関する間接サービスとしては，つぎのものがある：

- プログラム割込みエグジット・ルーチンの指示
- 何らかの事象とプログラムの実行の同期化
- 資源利用の逐次化
- 非同期的エグジット・ルーチンのスケジューリング

一方，監視プログラムの内的サービスには，つぎのものがある：

● タスク切換えの必要性のテストと表示
● ユーザー提供アドレスの妥当性テスト

以下でこれらサービスの機能構造について，説明してゆくことにしよう．

1. TCB に関する直接サービス機能

先述したように，処理プログラム用タスクはジョブ・ステップの実行の開始をきっかけに，開始／終止プログラムによって取り附けられる．しかしながら，ジョブ・ステップ用タスクだけがジョブを構成する唯一のタスクではない．ジョブ・ステップ用タスクの処理プログラムの中でサブタスクを生成することも許されている．

ジョブ・ステップ内に追加のタスクを生成する利点は，より多くのタスクでシステム資源の利用を競合させることにある．このような環境では，該当ジョブのどれかのタスクで待ち状態が生じたとき，制御権が移るのは必ずしも他のジョブのタスクとは限らない．該当ジョブの1部である別のタスクかも知れない．このことは，第4章の「プログラム構造と制御プログラムの働き」の「動的並列処理構造のプログラムと監視プログラムの働き」の個所で具体的に説明した．ジョブ・ステップ内に2つ以上のタスクを設けてジョブ・ステップの併列処理を行なうのは，2つ以上のタスクの間でかなりのオーバラップが期待できる場合である．ともあれ，ジョブ・ステップ内のタスクの相互関係を図式化すると**図8-26**のようになる．

図8-26の水平線はタスクをいくつかのレベルに分けている．これらレベルは，タスクの優先順位とは何ら関係しない．**親タスク**（originating task）と**サブタスク**（subtask）を識別しているに過ぎない．タスクA，B，A1，A2，A2a，B1およびB1aのいずれもジョブ・ステップ・タスクのサブタスクである．また，タスクA1，A2，A2aはタスクAのサブタスクである．一方，タスクA2aとB2aは，ジョブ・ステップ・タスクの最低位レベルのタスクである．タスクB1はタスクA1およびA2と同一レベルではあるが，タスクAのサブタスクとはみなされない．タスクAは，タスクA1とA2の親タ

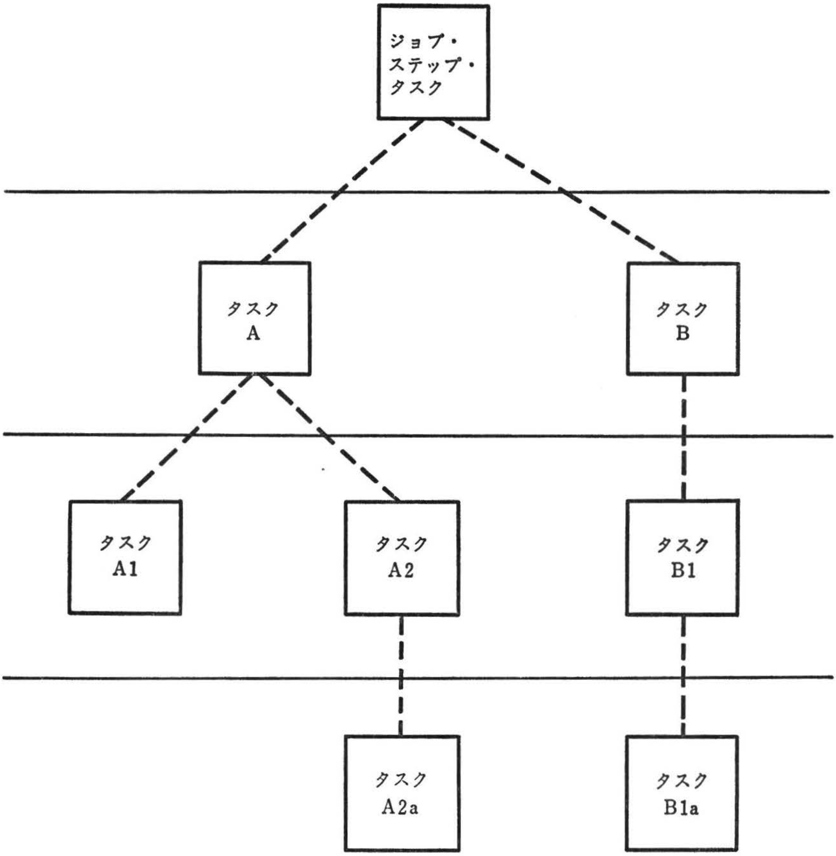


図 8-26　タスクの階層構造

スクであり，タスク A2 はタスク A2a の親タスクである．タスクの階層構造はジョブ・ステップ内に存在する．したがって，ジョブ・ステップ・タスク，タスクA，タスク A2 はタスク A2a の先輩にあたるが，タスクBはタスク A2a とは直接関係はないことになる．

さて，サブタスクの取り附けであるが，これは言うまでもなく ATTACH マクロ命令で行なわれる．サブタスクの取り附けとは，TCB の作成，作成した TCB への制御情報の挿入，サブタスクが必要とする主記憶装置の割当て，TCB 待行列に新規 TCB を入れること，さらにこの新しいサブタスクを実行するプ

表 8-1　ATTACH マクロ命令の一般形式

〔記号〕	ATTACH	{EP=記号名 | EPLOC=名前のアドレス | DE=リスト記入項目のアドレス} 〔,DCB=dcb アドレス〕 〔,LPMOD=数字〕〔,DPMOD=数字〕 〔,PARAM=(アドレス)〔,V1=1〕〕 〔,ECB=ecb アドレス〕〔,ETXR=エグジット・ルーチンのアドレス〕 〔,HIARCHY=数字〕 〔,GSPV=数字 | ,GSPL=リストのアドレス〕〔,SHSPV=数字 | ,SHSPL=リストのアドレス〕 〔,SZERO={YES | NO}〕 〔,STAI=(エグジットのアドレス〔,パラメータ・リストのアドレス〕) 〔,PURGE={QUIESCE | HALT | NONE}〕〔,ASYNCH={YES | NO}〕〕 〔,TASKLIB=dcb アドレス〕

ログラムを得るために内容監視プログラムとの連係を計ることを意味する．新しいサブタスクが作動可能状態の TCB のうちで最高の指名順位にあるなら，指定されたプログラムに直ちに制御権が渡される．そうでないなら，制御権の委譲は一時待たされることになる．

呼び出しプログラムでは，CHAP マクロ命令を用いてサブタスク TCB の指名順位の変更を要求することもできる．CHAP ルーチンは，新しい指名順位を計算し，その結果の妥当性を検査し，当該 TCB にその指名順位の値を挿入し，TCB 待行列の妥当な位置に TCB を入れ変え，さらに呼び出しプログラムの代りに指名順位が変更された TCB のプログラム・ルーチンに制御権を受け渡すべきかどうかをテストする．これが，CHAPルーチンの機能である．

また，TCB から特定の情報を取りだすこともできる．この場合，TCB は呼び出しプログラムの TCB であっても，サブタスクの TCB であってもよい．特定情報は EXTRACT マクロ命令で得られるのであるが，この命令のオペランドで指定した回答域に要求した制御情報が入れられる．

表 8-2 ATTACH マクロ命令のパラメーター一覧表

パラメータ	指定の内容	意　　味
EP=	記号名	制御権を受渡すロード・モジュールの入口点名.
EPLOC=	名前のアドレス	入口点名が貯えられている主記憶域のアドレス.
DE=	リスト記入項目のアドレス	入口点名に対するリスト記入項目の名前欄のアドレス. リスト記入項目は BLDL マクロ命令で作られるものである.
DCB=	dcb アドレス	上述の入口点名を持ったロード・モジュールが収容されている区分データ・セットの dcb アドレス. 私的ライブラリーに収容されているとき指定.
LPMOD=	数字	親タスクの限界優先順位から減算すべき数. この結果が新しいタスクの限界優先順位となる. 省略すると親タスクの限界優先順位が与えられる.
DPMOD=	数字（符号つき）	親タスクの指名順位に代数加算すべき符号つきの数. この結果が新しいタスクの指名順位となる(ただし, 限界優先順位をこえないかぎり). 省略すると, 新しいタスクの限界優先順位と親タスクの指名順位のうちどちらか小さい数が与えられる.
PARM=	アドレス	呼び出しプログラムに受け渡すべきアドレス・パラメータをカンマで区切って指定. 制御権が渡ると, レジスター1に最初のパラメータのアドレスが入る.
VL=	1	受け渡すパラメータの数が可変であるとき指定. この指定があると, 最後のアドレス・パラメータの最高位ビットが1にセットされる.
ECB=	ecb アドレス	制御プログラムが新しいタスクの終了状態を示すのに使う ECB のアドレス. このパラメータを指定すると, サブタスクが終了したあとシステムからこのサブタスクを除去するためには DETACH マクロ命令を使用する必要がある.
ETXR=	エグジット・ルーチンのアドレス	新しいタスクが正常に, ないしは異状に終了した場合に制御権を渡すべきタスクの終りのエグジット・ルーチンのアドレス. このパラメータの指定があると, サブタスクが終了したあとシ

パラメータ	指定の内容	意　味
		ステムからこのサブタスクを除去するためにはDETACH マクロ命令を出す必要がある.
HIARCHY=	数字(0または1)	主記憶装置のハイアラキー・サポートがある場合に，ロード・モジュールをローディングすべき主記憶装置のハイアラキーを指定.
GSPV=	数字	主記憶域のサブプール番号を指定. 指定されたサブプールの所有権は新しいタスクに割り振られ，もはや親タスクのプログラムではその記憶域は使えない.
GSPL=	リストのアドレス	主記憶域のサブプール番号のリストのアドレス. 指定されたサブプールの所有権は新しいタスクに割り振られ，もはや親タスクのプログラムではその記憶域は使えない.
SHSPV=	数字	主記憶域のサブプール番号を指定. 親タスクと新しいタスクの双方のプログラムでこの記憶域を利用できる.
SHSPL=	リストのアドレス	主記憶域のサブプール番号のリストのアドレス. 親タスクと新しいタスクの双方のプログラムでこの記憶域を利用できる.
SZERO=	YES または NO	サブプール0をサブタスクとの間で共有できるかどうかを指定.
STAI=	エグジットのアドレス	ATTACH アクロ命令を出したタスクのサブタスクのいずれかが異常終止したときに制御権を渡すエグジット・ルーチンのアドレス. このエグジット・ルーチンには，新たに作られたサブタスクで生成した下位レベルのサブタスクが異常終止してエラー回復の見込みがない場合にも，制御権が渡される.
	パラメータ・リストのアドレス	STAI エグジット・ルーチンで必要となるパラメータ・リストのアドレス.
PURGE=	QUIESCE	指定されたSTAIエグジットがとられたとき，未処理の入出力オペレーション要求を保持することを示す. STAI エグジット・ルーチンの終りに，その入出力オペレーションを処理する再試行ルーチンをコーディングすることができる.

表 8-2 ATTACH マクロ命令のパラメーター一覧表（つづき）

パラメータ	指定の内容	意　味
	HALT	指定されたSTAIエグジットがとられたとき，未処理の入出力オペレーション要求を保持しないことを示す．
	NONE	STAI エグジットがとられたとき，入出力処理が正常に続行されるようにすることを示す．
ASYNCH=	YES	非同期的な割込み処理のために，STAI エグジット・ルーチンでの処理を中断させるようにすることを示す．
	NO	非同期的な割込み処理のために，STAI エグジット・ルーチンでの処理を中断させるようなことはないことを示す．
TASKLIB=	dcb アドレス	ジョブ・ライブラリーに対応するオープンされた DCB のアドレス．このライブラリーのことを**タスク・ライブラリー**と呼ぶ．このタスク・ライブラリーが，ATTACH されたモジュールの入口点名とサブタスクでアクセスされる後続モジュールの入口点名の双方を探索するのに用いられる．

TCB のサブタスク待行列からサブタスク TCB を取り除く仕事が，サブタスクを終了させるための最終ステップである．DETACH ルーチンでは，TCB を除去するだけでなく，そのサブタスクに属していた主記憶域をも解放する．この主記憶域には，サブタスク TCB そのものが占めていた記憶域あるいはそれに関連したプログラムのレジスター保存域なども含まれる．

ともあれ，TCB に関するこれら 4 つの 直接サービスの機能について 以下で説明することにしよう．

1) サブタスクの取り付け機能

ユーザー・プログラムあるいはシステム・ルーチンで ATTACH マクロ命令を出すと，監視プログラムは呼び出しタスクのサブタスクとして ATTACH マクロ命令で指示したプログラムの実行を開始するようにする．この ATTACH マクロ命令の形式とパラメータの内容をまとめたのが **表 8-1** と **表 8-2** であ

る.

ATTACH マクロ命令は，監視プログラムに新しいタスクを生成することを指示し，その新タスクが**活動状態**（active）になったとき制御権を渡すべきプログラムの入口点を指示する．指定する**入口点名**（entry point name）は区分データ・セットの登録簿にあるメンバー名か別名，あるいはあらかじめ IDENTIFY マクロ命令で指示した名前である．指定した入口点名をつきとめることができないと，当然のことながら新しいサブタスクは異常終止されてしまう．

新タスクの TCB アドレスは汎用レジスター1に戻される．新しいタスクは，**親タスク**（originating task）のサブタスクである（この場合の親タスクとは，ATTACH マクロ命令が出されたとき活動状態にあったタスクである)．サブタスクの限界優先順位と指名順位とは，ATTACH マクロ命令で指示されないと親タスクのそれと同じものになる．

制御権を受渡すロード・モジュールは，主記憶装置に利用可能なコピーが存在しない場合には，呼び込みが行なわれる．なお，新タスクの終了状態を知らせるための ECB，新タスクが終了したときに制御権を渡すエグジット・ルーチン，あるいは新タスクに受渡すべきパラメータのアドレスなどを ATTACH マクロ命令で与えることもできる．ECB または ETXR オペランドを指定すると，システムからサブタスクを取り除くために DETACH マクロ命令を出す必要がある．一方，これらのオペランドを省略した場合には，サブタスクの処理が終了すると自動的にシステムから取り除かれる．ATTACH マクロ命令では，さらに，サブプールの所有権を新タスクに割り当てるか，または親タスクとの間で共用するようにするかの指定もできる．これらオペランド・パラメータの詳細に関しては，**表 8-2** を参照されたい．

ATTACH マクロ命令は，SVC 割込みをひきおこす．その結果，割込み処理ルーチンは，ATTACH SVC ルーチンに制御権を与え，この SVC ルーチンで要求されたサービス機能を果たす．ATTACH SVC ルーチンの主要な機能には，つぎのようなものがある：

- 新しい TCB 用の主記憶域を得る．

- サブタスクを制御するのに必要な情報を新しい TCB に入れる.
- 親タスクに属していたサブプールをサブタスクに対して割当てる.
- 新 TCB のアドレスを2種類の待行列（親タスクのサブタスク待行列とディスパッチャーが使用する TCB 待行列）に入れる.
- 新しいサブタスクを実行するのに必要な最初のプログラムをつきとめ，必要ならそのプログラムを呼び込み，実行をスケジュールするために，内容監視プログラムを連係する.

ATTACH ルーチンは，以上のような主要な機能を果たすと同時につぎのような機能も達成してくれる：

- ATTACH マクロ命令で ETXR オペランドが指定されているとき，**タスク終了エグジット・ルーチン**（end-of-task exit routine）をスケジューリングし制御するために必要となる制御ブロック（IQE と IRB）のための主記憶域を得る.
- その IQE と IRB に制御情報を入れる.

作成された TCB は，ATTACH ルーチンによって初期値化され，タスクの状況に関する情報を入れ，そのサブタスクを実行するプログラムで必要となる各種待行列の起点アドレスを所定のフィールドに入れてくれる．たとえば，TCB に対応した現行プログラムを表示するのには，RB 待行列上の最上位の RB をさすポインター・アドレスを TCB の所定欄に挿入してくれる.

つぎに ATTACH SVC ルーチンの機能を順を追って説明しよう：

① TCB，IRB，IQE のための主記憶域の確保

確保する主記憶域は，新しいサブタスク用 TCB およびサブタスク用のタスク終了エグジット・ルーチンをスケジュールし制御するために用いる，1種類ないしは2種類の制御ブロックのためのスペースである．この制御ブロックとは，タスク終了エグジット・ルーチンの実行を制御する**割込み要求ブロック**（Interruption Request Block；略して **IRB**）と，エグジット・ルーチンのスケジュールを援助する**割込み待行列要素**（Interruption Queue Element；略して

IQE）のことである．

ATTACH ルーチンは，これら制御ブロックを記憶するスペースを**システム待合せ域**（System Queue Area；略して **SQA**）から確保する．SQA は，サブプール 253 である．ところで，作成すべき制御ブロックはATTACH マクロ命令のパラメータの与え方によって異なる．したがって，それに応じて SQA（サブプール 253）から取得するスペースも異なる．つまり，ATTACH マクロ命令で ETXR オペランドが指定されていないと，TCB 用のスペースだけがとられる．ETXR オペランドが指定されていて，かつエグジット・ルーチンのための IRB がすでに用意されている場合だと，TCB と IQE のためのスペースが確保される．しかし，ETXR オペランドが指定されていて，エグジット・ルーチンのための IRB がまだ作られていないと，TCB, IQE に加えて IRB 用のスペースもとられる．

先述したようにタスク終了エグジット・ルーチンを実行するには，IQE と IRB が必要である．ただし，IRB は1つのエグジット・ルーチンに1個だけ用意されていればよい．複数個のサブタスクで同一のエグジット・ルーチンを使用するときは1つの IRB があればよい．そこで，ATTACH マクロ命令で ETXR オペランドが指定されている場合には，すでに IRB が存在するかどうか判定し，その結果にもとづいて IRB 用のスペースを確保すべきか否かを決定するのである．**図 8-27** に示しているように，全部のサブタスクのエグジット・ルーチン用の IRB を調べ，同じ入口点を待った IRB が存在するかどうかで，その有無を判定している．

② IQE, IRB, TCB の初期値化

ATTACH SVC ルーチンは，IQE 用の記憶域を確保すると，その各フィールドを**表 8-3** で示しているように初期値化する．IQE を初期値化するのに加えて，ATTACH ルーチンは，IRB の中の“USE”カウント（RBUSE フィールド）を1だけ増やす．この “USE” カウントは，同じ IRB を使用するサブタスクの個数を表示している．監視プログラムのエグジット・ルーチンでは，タスク終了ルーチンの実行が完了するつどこのカウントを1ずつ減ずる．したがっ

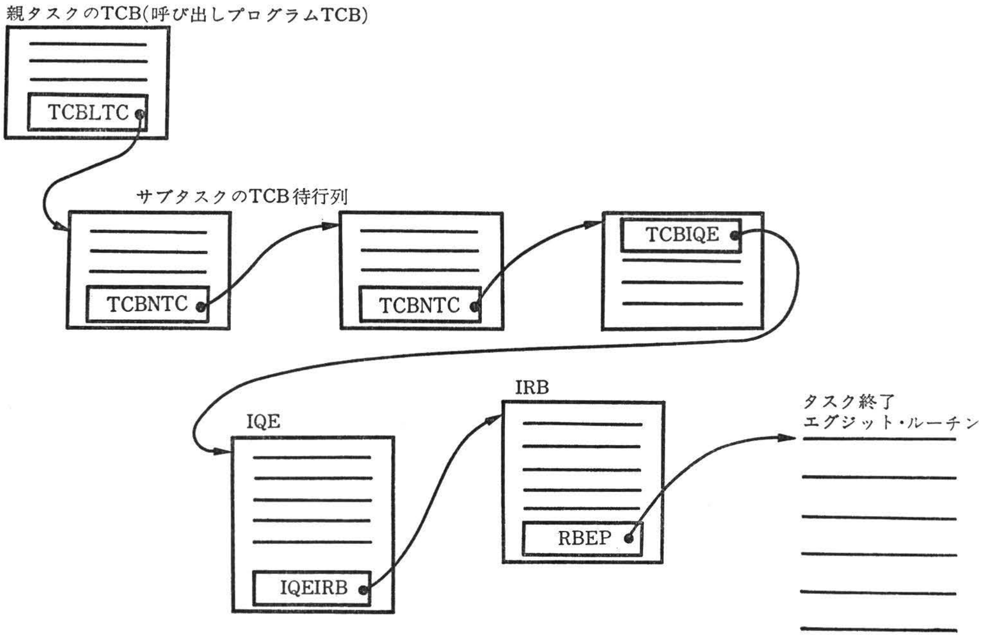


図 8-27　TCB, IQE, IRB およびタスク終了エグジット・ルーチンの相互関係

て，カウントが0になると，監視プログラムのエグジット・ルーチンはIRBで占有されていた主記憶域のスペースを解放してくれる．

さらに，ATTACH ルーチンは新しく作りあげたサブタスク TCB のレジスター保存域を除いた全フィールドをまずクリアーし，そのあとで TCB の各フィールドに必要な情報を入れることによって TCB の初期値化をはかる．ATTACH マクロ命令で ETXR オペランドが指定されていると，IQE のアドレスを TCB の TCBIQE フィールドに挿入し，タスクの終了エグジット・ルーチンの要求がなされた旨の標識をセットする．

③ 親タスクの TCB からサブタスク TCB への情報の移動

親規作成のサブタスク TCB を初期値化したあとで，ATTACH ルーチンはその TCB のある種のフィールドへ親タスク (caller's task) から情報を移動する．ある種のフィールドとは，ジョブ・ステップ内の全ての TCB の中で同

表 8-3　IQE を構成するフィールドと初期値化

フィールド名	フィールド内の情報タイプ	フィールドの初期値化
IQELINK	IQE 待行列の中の次の IQE のアドレス	ゼロにする
IQEPARAM	タスク終了エグジット・ルーチンに受渡すパラメータ	新たに作成されたサブタスクの TCB アドレスを入れる
IQEIRB	IRB のアドレス	直前に作りあげた IRB のアドレスまたはサブタスク待行列を探索中に見つかった IRB のアドレスを入れる
IQETCB	TCB のアドレス	呼び出しプログラムのTCB（つまり，親タスクのTCB）のアドレスを入れる

じ情報をもつ必要のあるフィールドのことである．つまり，これらフィールドには，区分待行列へのポインター（TCBPQE），タスク入出力テーブルへのポインター（TCBTIO），ジョブ・ライブラリー用 TCB へのポインター（TCB-JLB）が含まれる．また，**システム管理機能**（System Management Facility；略して **SMF**）がシステムにもたされている場合には，時間制御テーブルへのポインター（TCBTCT）にも情報が移動される．なお，ATTACH マクロ命令で TASKLIB＝dcb アドレスを指示していると，TCBJLB フィールドへは親タスクの TCB からの情報移動を行なわない．その代り，TASKLIB で指定した dcb アドレスをこのフィールドへ挿入する．

④　サブタスク TCB フィールドへのパラメータ情報のセット

親タスク TCB からの不変情報をサブタスク TCB の所定フィールドへ移動する仕事が終ると，つぎは ATTACH マクロ命令で指定したパラメータ情報をサブタスク TCB にセットする．これには，下記情報と初期値化が含まれる：

- ECB のアドレス（ATTACH マクロ命令での指定があれば）のセット
- サブタスクの制限優先順位と指名順位のセット

- ロールアウト資格フィールド（TCBNROC）のセット
- TCBFRA フラグのセット
- TCBJSTCB フィールドの初期値化
- TCBJPQ フィールドの初期値化
- TCBPKF フィールドの初期値化（必要なら）
- TCBFSA フィールドの初期値化（必要なら）
- TCBMSS フィールドの初期値化（必要なら）
- TCBJSCB フィールドのアドレスのセット

このあと，ATTACH ルーチンは入力パラメータから，制限優先順位と指名順位を決定し，TCB の所定欄に貯える．サブタスクの制限優先順位は入力パラメータにもとずいてセットされるが，親タスクのそれを越えないようにする．同様に指名順位についても，入力パラメータに準拠するとはいうものの，サブタスク自身の制限順位を越えないように調整する．これら順位が ATTACH マクロ命令で指定されていないと，親タスクの制限優先順位と指名順位に等しい値がサブタスクにも適用される．なお，ATTACH マクロ命令が，タイムシェアリング用タスクから出された場合には，実際の制限優先順位と指名順位およびタイムシェアリング用の制限優先順位と指名順位とが，サブタスク TCB のそれぞれの対応欄に移される．

⑤ サブタスクへのサブプールの割当て

ATTACH ルーチンは，ATTACH マクロ命令で指定したパラメータに応じて，サブタスク TCB のプログラムに主記憶装置のサブプールの割り当てを行なう．**GSPV** ないしは **GSPL パラメータ**（総称して **"give" パラメータ**というが）が与えられると，指示されたサブプールが排他的利用用のサブプールとして，サブタスクのプログラムに割り当てられる．したがって，割り当てられたサブプールはもはや親タスクでは使用できなくなる．一方，**SHSPV** ないしは **SHSPL パラメータ**（総称して **"share" パラメータ**というが）与えられると，サブタスクのプログラムと親タスクのプログラムとで同じサブプールをアクセスすることができるようになる（**図 8-28** 参照）．

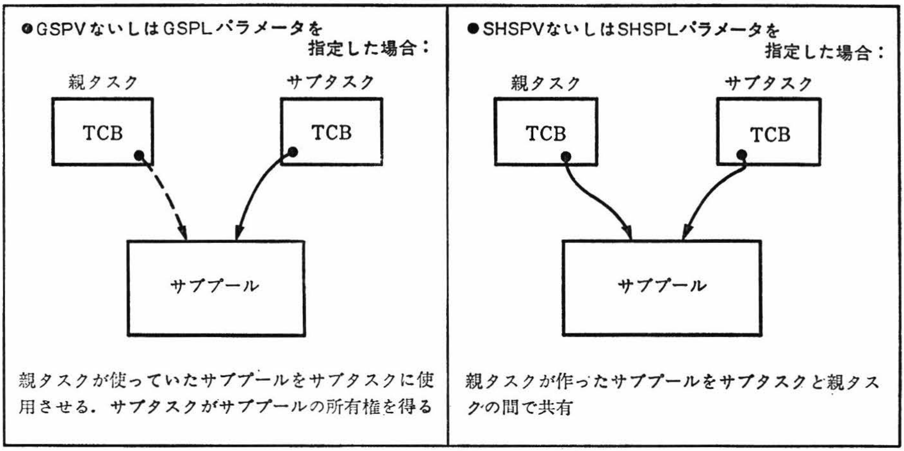


図 8-28　サブプールの割当て

"give" パラメータが指定されていると，ATTACH ルーチンは親タスクの SPQE 待行列を探索し，"give" するサブプールに対応する SPQE が発見されるとそれを親タスクの SPQE 待行列から取り除くと同時にサブタスク TCB のサブプール待行列に入れる．対応する SPQE が存在しない場合には，新しい SPQE を作って "owned" のフラグを立て，サブタスク TCB のサブプール待行列に入れることになる．

"share" パラメータの指定があれば，やはり親タスクの SPQE 待行列を探し，"share" するサブプールに対応する SPQE がある時は，サブタスク用の SPQE を新たに作り，"shared" のフラグを立てたうえで，サブタスク TCB 用のサブプール待行列に入れる．対応する SPQE が存在しない場合には，親タスク用の SPQE とサブタスク用の SPQE の2種類を作る．親タスクのための SPQE には "owned" と "shared" の両方のフラグを立て，親タスク用の SPQE 待行列に入れる．一方，サブタスクのための SPQE には "shared" のフラグを立てて，サブタスク TCB の SPQE 待行列に入れることになる．

サブプール0は主記憶装置の特殊なサブプールであり，上述の方法とは別に処理される．したがって，"give" ないしは "share" パラメータでサブプール

0を指定しても，その指定は無視される．ATTACH ルーチンでは，SZERO パラメータを検査し，サブプール0を共有すべきかどうか決定する．共有しない場合には，サブタスク用の主記憶装置の割当てを完了し，つぎの機能に移る．しかし，サブプール0を共有することになっていると，親タスクのサブプール待行列を探索し，必要なサブプール待行列要素（SPQE）を作りあげたあと，親タスク TCB あるいはサブタスク TCB からチェイニングするようにする．

⑥ 待行列へのサブタスク TCB の配置

システム内でのタスクの階層構造を図示すると，**図 8-29** のようになる．すでに説明したように，マスター・スケジューラー用タスクがシステム制御用タスクを Attach する．システム制御用タスクは，条件に応じて読取プログラム用タスク，開始プログラム用タスク，書き出しプログラム用タスクのいずれかを取り附ける．このうち開始プログラム用タスクがジョブ・ステップ用タスクを Attach する．ジョブ・ステップ用タスクでは，必要に応じて ATTACH マクロ命令を用いてサブタスクを取り附ける．

ところで，Attach ルーチンは取り附けたサブタスク用の新規 TCB を2種類の待行列上に配置する．この2種類の待行列とは，親タスク用のサブタスク待行列と TCB 待行列のことである．サブタスク待行列は，TCB が作られた順序を示すものであり，サブタスクが生成された順にならべられる．監視プログラムの中の ABEND ルーチンは，異常終了のさいにこの待行列を使ってジョブ・ステップで用いていた諸資源を解放する順序を決定する．一方，TCB 待行列は優先順位順にならべられた TCB の待行列である．後述する CHAP SVC ルーチンがこの待行列を用いて，TCB の指名順序を変更する．また，ディスパッチャー・ルーチンはつぎに実行すべきタスクを決定するのに，この待行列を用いる．つまり，TCB 待行列を上から下へと検索し，最も優先順位の高い作動可能状態の TCB (ready TCB) を決定する．

図 8-30 で示しているように，サブタスク待行列と TCB 待行列とは，物理的には全く同じ TCB から構成されている．各 TCB 内の異ったポインターを用いて待行列が構成され，処理されるに過ぎない．TCB 待行列は TCBTCB フ

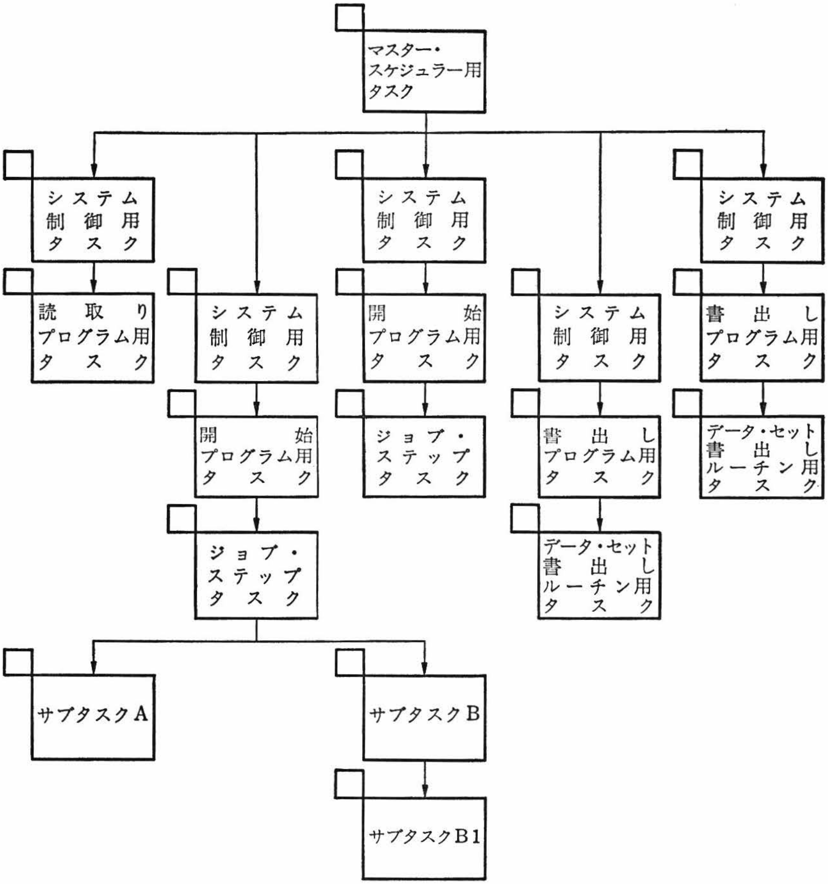


図 8-29　システム内のタスクの階層構造

ィールドを用いて，指名順位の高い順に構成される．サブタスク待行列は TCBNTC と TCBLTC フィールドを用いて，サブタスクの生成順に構成される．図8-30では，サブタスクA，サブタスクB，サブタスクB1の順になる．

⑦　タスク・ディスパッチとプログラム・フェッチの準備

親タスクと取り附けられたサブタスクとはあたかも独立したタスクのごとく実行される必要がある．そのために，ATTACH SVC ルーチンは図8-31のよ

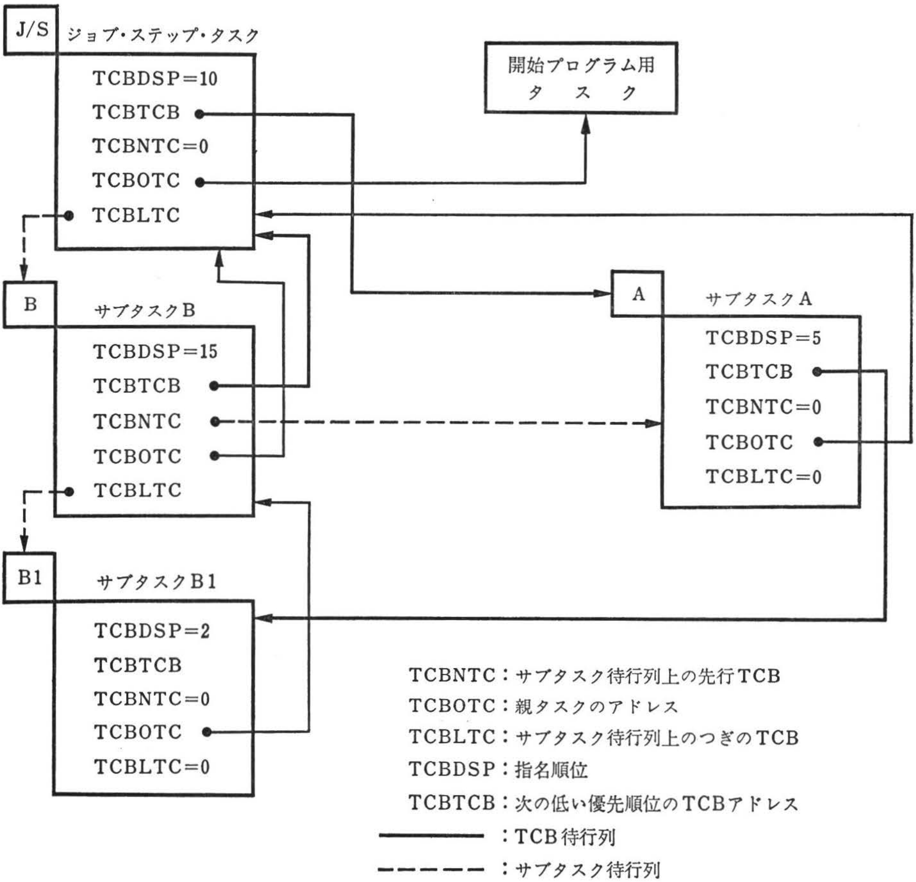


図 8-30　TCB 待行列とサブタスク待行列

うに制御ブロックの連鎖関係を変更する．つまり，ATTACH SVC ルーチンの SVRB を親タスクから取り除き，サブタスク TCB の待行列とするのである．

ATTACH ルーチンは親タスクに制御権を戻すための準備をあらかじめ行なっておく．**第二次 SVC 割込み処理ルーチン** (SVC Second Level Interruption Handler) が SVRB に貯えていた親タスクのレジスターの内容を親タスク TCB のレジスター保存域に移す．また，新タスク TCB の TCBGRS (汎用レジスターの保存域) に新しいサブタスク TCB のアドレスも貯える．これらの情報

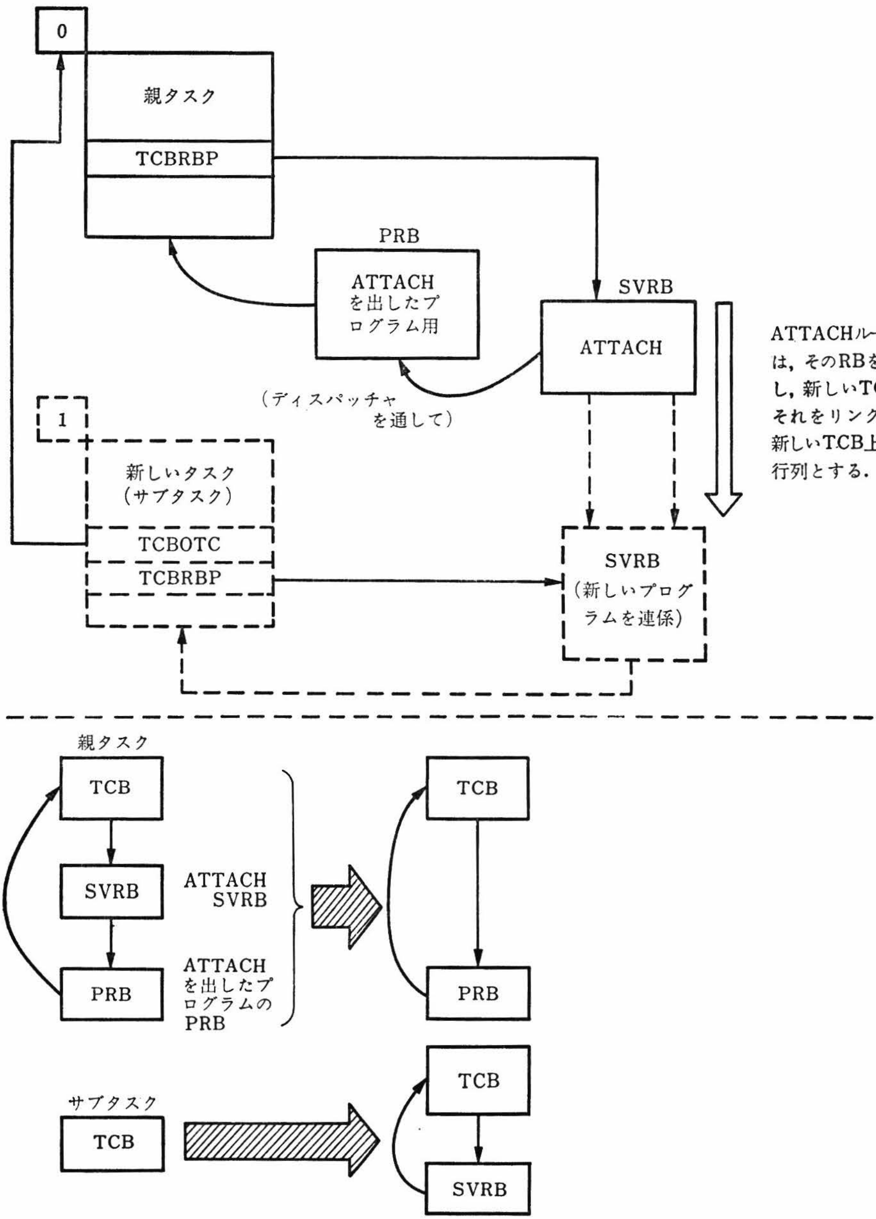


図 8-31　ATTACH 処理構造

は，親タスクに制御権が戻されるときに，ディスパッチャーが汎用レジスターにロードするのである．**取り付けプログラム**(attaching program)，つまり親タスクが再度ディスパッチされると，ATTACH マクロ命令の直後の命令に制御権が渡る．このさい，レジスター1にある新しいサブタスク TCB のアドレスが ATTACH ルーチンから引き渡されるパラメータということになる．ともあれ，SVRB の連鎖関係が変更されるさいに，SVRB に保存されていた汎用レジスターの内容が親タスク TCB の TCBGRS に移されるのであり，これで親タスクのディスパッチの準備は完了することになる．

一方，ATTACH マクロ命令で指示されたプログラムの実行を保障するためには，内容監視プログラムの中の LINK 機能の助けをかりる．サブタスクに連鎖された SVRB の旧 PSW の命令アドレスが LINK ルーチンのアドレスをポイントしている．したがって，サブタスクがディスパッチされると SVRB が現行 RB となるので，必然的に LINK 機能が果される仕組みとなっている．LINK ルーチンは，所要のプログラムが主記憶域ないしはプログラム・ライブラリーにあるのをつきとめ，必要に応じて該当プログラムを主記憶域にフェッチし，新規に生成したサブタスク用のプログラムとして連係をはかる．

このようにして，親タスクとサブタスクのディスパッチの準備を完了し，ATACH SVC ルーチンからディスパッチャーにブランチすることになる．以上みてきたようなステップをふんで，サブタスクの取り付け機能を遂行するのが ATTACH SVC ルーチンである．

2) タスクの指名順位の変更機能

CHAP SVC ルーチンは，問題処理プログラムまたはシステム・プログラムに対してそれ自身のないしはサブタスク TCB の指名順位の変更を実現可能にする．サブタスク TCB は，当然のことながら CHAP マクロ命令を出したタスクのサブタスクであることが前提となる．つまり，親タスクのプログラムで取り付けられ，そのサブタスク待行列に収容されていることが必要である．CHAP マクロ命令を出したプログラムでは，ゼロとその TCB の**制限優先順位**(limit priority）の間なら，指定した TCB の指名順位をどのような値にでも

表 8-4　CHAP マクロ命令の一般形式

〔記号〕	CHAP	指名順位変更値 [, tcb のロケーション・アドレス / , 's']

指名順位変更値：指示されたタスクの指名順位に加算すべき符号付き数値.

tcb のロケーション・アドレス／'s'：活動状態のタスク(親タスク)のサブタスク用 TCB のアドレスを記憶している1ワードの記憶域のアドレス. tcb のロケーション・アドレスの代りに 's' を指定すると，活動状態のタスク(親タスク)の優先順位を変更することを示す.

変更することができる．なお，CHAP マクロ命令の一般形式は**表 8-4** で示した通りである．

指名順位と制限優先順位は，いずれも ATTACH マクロ命令のパラメータで指定できるものであるが，両者ははっきりと異った機能を果たす．指名順位は，TCB 待行列内の TCB の位置を決定する．換言すれば，割込み後に実行すべきプログラム・ルーチンの順序を示している．ディスパッチャーは，最高の指名順位をもった作動可能状態の TCB に属するプログラムを実行できるようにスケジュールする．一方，制限優先順位は CHAP SVC ルーチンが利用するためのものであり，TCB の指名順位をあげる場合の制限値を指示している．

時分割機能（time-sliced feature）が適用されていると，CHAP ルーチンは指定されている TCB が時分割タスクであるかどうかテストする．このために，TCB の中の TCBFLGS フィールドの時分割ビットが調べられる．このビットが１にセットされていると時分割タスクである．CHAPルーチンはこのビットをゼロにしてから，このタスクの指名順位に対応する**時分割制御要素**（Time-Sliced Control Element；略して **TSCE**）を見つけだす（**図 8-32** 参照）．該当 TCB のアドレスが TSCE の最初の TCB，最後の TCB あるいはつぎの TCB フィールドのいずれにも一致しないと，新らたな指名順位が決定される．しかも，TSCE には何らの変更も加えられない．CHAP マクロ命令で指示された TCB のアドレスが TSCE のいずれかのアドレスと同じなら，CHAP ルー

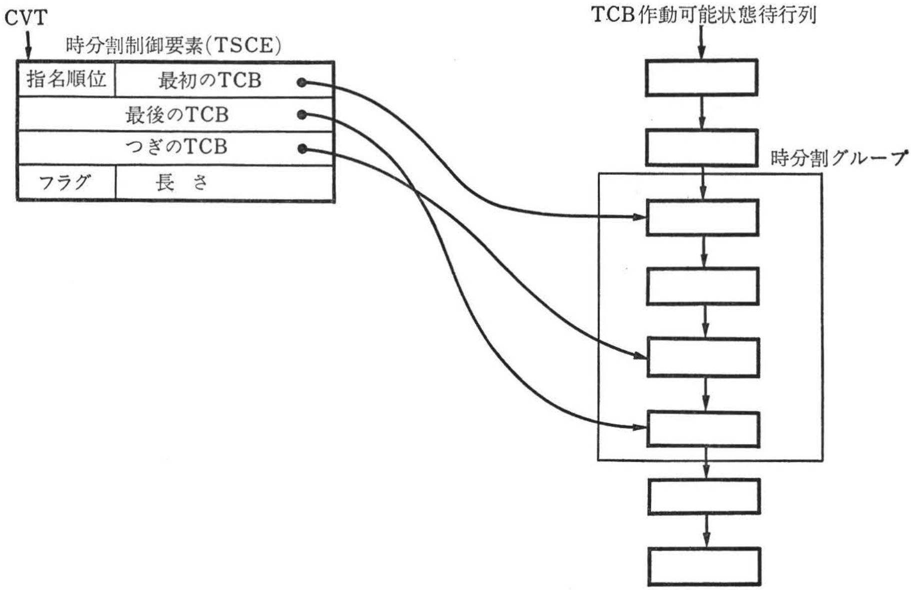


図 8-32 時分割制御要素と時分割グループの関係

チンは所定のポインターを変更することになる.

一方, タイムシェアリング用タスクだと, 通常の指名順位と制限優先順位の代りにタイムシェアリング用タスクの指名順位 (TCBTSDP フィールド) と制限優先順位 (TCBTSLD フィールド) を用いる. ユーザー・サブグループの中のタスクだけが CHAP の影響をうける. 変更された TCB は, TCB 待行列でつぎの指名順位をもったタスクの直前に入れられる.

すべての妥当性検査が終ると, CHAP ルーチンはマクロ命令で指示されたタスクの指名順位を変更し, それがゼロ以下になっていないかどうか, マクロ命令を出した**活動状態のタスク** (active task) の制限優先順位を越えていないかどうかを確認する. また, サブタスクの指名順位がそのサブタスクの制限優先順位を越えた値に変更された場合には, その制限優先順位を新しい指名順位と同じ値に変える.

時分割機能が採用されていると, 新しい指名順位が時分割であるかどうか再テストする. CHAP ルーチンは, その指名順位に対応する TSCE をみつける

と，TCB の中の時分割ビットをセットする．そのあとで，TSCE の“つぎのTCB”フィールドを調べる．そのフィールドがゼロであると，該当タスクが時分割グループの唯一のメンバーであるので，TSCE の“最初の TCB”，“最後の TCB”，“つぎの TCB”の各フィールドに当該 TCB のアドレスを挿入する．“つぎの TCB”フィールドがゼロでないと，当該 TCB のアドレスが“最後のTCB”フィールドに貯えられる．

話をもとに戻そう．TCB の指名順位が変更されたとなると，つぎに CHAP ルーチンは TCB 待行列を再調整し，指名順位の高い順に並びかえる．CHAP ルーチンは，TCB 待行列を探索中に**タスク切換えルーチン**（task switch routine）にブランチし，指名順位が呼び出しタスクのそれよりも高い作動可能状態のタスクがあるかどうか判定する．このような状態は，つぎの2つの場合に生じる．つまり，呼び出しタスクがサブタスクの指名順位を自分の指名順位よりも高いものに変更した場合，または逆に自分の指名順位をサブタスクのそれより低い値に変えてしまった場合である．このような場合には，TCB 待行列の順序が調整されると，呼び出しタスクよりも低い優先順位をもっていた TCB が今や呼び出しタスクの指名順位をしのいでしまうのである．なお，**多重処理システム**（multiprocessing system）の環境下では，2番目の CPU で実行中のタスクの指名順位よりも高い順位をもった作動可能状態の TCB がでてくることがあることもうなづけよう．

ともあれ，タスク切換えルーチンが活動中のタスクよりも高い指名順位をもった作動可能状態のタスクをみつけると，ディスパッチャーに対してタスク切換えの必要がある旨の表示をする．この表示は，高い指名順位の TCB アドレスを新 TCB ポインターの場所に貯えることによって達成される．ディスパッチャーは，この新 TCB ポインターを点検することで，割込みを起したルーチンを再ディスパッチすべきなのか，別の作動可能状態のタスクに属するルーチンをディスパッチすべきかを決定するのである．

3）TCB 中の特別情報の取出し機能

EXTRACT マクロ命令を与えることによって，TCB の中の特定の情報およ

びそのタスクに従属する JFCB からの特定情報を得ることができる. EXTRACT マクロ命令の一般形式は**表 8-5** に示した通りである. TCB の中の汎用レジスター保存域のアドレス, 浮動小数点レジスター保存域のアドレス, タスク出口ルーチンの終りのアドレス, 制限優先順位と指名順位, 完了コード, タスク入出力テーブルのアドレスなどを得ることができるのである.

EXRTACT マクロ命令を与えることによって, 監視プログラム呼び出し割り込みが生じ, EXTRACT SVC ルーチンが機能してくれる. このルーチンの目的は, 上述したように, 指定したタスクの TCB や JFCB からの特定情報を呼び出しプログラムに与えることにある. EXTRACT マクロ命令で指定できる TCB は, 活動状態のタスク (EXTRACT マクロ命令を出したプログラムのタスク) の TCB, またはそのプログラムで取り附けられたサブタスクの TCB のいずれかである. 表 8-5 でも示しているように, 入手することのできる情報は全フィールドであってもよいし, 任意のフィールドの組み合わせであってもよい. EXTRACT SVC ルーチンは, 取出した情報を EXTRACT マクロ命令で指定した回答域に入れてくれる. この様子は, **図 8-33** で図示している.

EXTRACT SVC ルーチンは, 単に要求された情報を回答域に戻してくれるだけではない. 問題プログラムに受渡すパラメータ (情報) が妥当なものであるかどうかを決定するための検査も行なう. この妥当性検査によって, 無意味なデータの取り出しや解釈困難な原因のプログラム割込みを未然に防止している. 妥当でないパラメータの指定があると, SVC ルーチンで誤りコードをセットしたうえでそのタスクを異常終止させる仕組みになっているのである.

4) サブタスクの消去機能

ATTACH マクロ命令で ECB または ETXR オペランドを指定してサブタスクを生成した場合には, そのサブタスクをシステムから消去するために DETACH マクロ命令を使用する必要がある. ECB または ETXR オペランドを指定したということは, サブタスクの終了状態を親タスクで検査することを暗示している. ともあれ, このような方法で生成したサブタスクは, 親タスクの実行を終了する前にあらかじめシステム内から消去してしまう必要がある. サ

表 8-5　EXTRACT マクロ命令の一般形式

〔記号〕	EXTRACT	回答域のアドレス〔, tcbのローケーション・アドレス / , 's'〕, FIELDS=(コード)

回答域のアドレス：ワード境界からはじまる主記憶域のアドレス．確保すべきワードの個数は，FIELDSオペランドで指示したコードの個数に一致する．ただし，FIELDS=(ALL)とすると7ワードの大きさが必要である．

tcbのローケーション・アドレス：活動状態のタスクのサブタスク TCB のアドレスを記憶している1ワードの記憶域のアドレス．tcb のローケーション・アドレスの代りに 's' を指定すると，活動状態のタスクの TCB から情報を得ることを示す．

コード：下記コードをカンマで区切って記入する．要求フィールドの情報は，図8-33に示した形式で戻される．

コード	要求内容
GRS	タスクが活動状態でないときに，制御プログラムが汎用レジスターの内容を保存するのに用いる保存域 (TCBGRS) のアドレス．
FRS	タスクが活動状態でないときに，制御プログラムが浮動小数点レジスターの内容を保存するのに用いる保存域 (TCBFRS) のアドレス．
AETX	ATTACH マクロ命令で指定したタスク出口ルーチンの終りのアドレス．
PRI	そのタスクの制限優先順位と指名順位．
CMC	そのタスクの**完了コード** (completion code).
TIOT	タスク入出力テーブルのアドレス．
COMM	コマンド・スケジューラーのコミュニケーション・リストのアドレス．
PSB	JSCB から取り出される**保護記憶制御ブロック** (protected storage control block ; 略して **PSCB**) のアドレス．タイムシェアリングのときのみ有効．
TSO	TCB のタイムシェアリング標識フィールド (TCBTSFLG) のアドレス．タイムシェアリングのときのみ有効．
TJID	tcb のローケーション・アドレスで指定したタスクのターミナル・ジョブ標識．タイムシェアリングのときのみ有効．
ALL	GRS, FRS, AETX, PRI, CMC および TIOT フィールドからの情報を要求．

回答域のアドレス

	1バイト	1バイト	1バイト	1バイト	
GRS	アドレス				ALL
FRS	アドレス				ALL
	予約フィールド（ゼロにセット）				ALL
AETX	アドレス				ALL
PRI	0 0	0 0	制限優先順位	指名順位	ALL
CMC	タスク完了コード				ALL
TIOT	アドレス				ALL
COMM	アドレス				
TSO	アドレス				
PSB	アドレス				
TJID	標識				

（全体で4バイト）

図 8-33　EXTRACT 回答域のフィールド形式

ブタスクの消去に失敗すると，親タスクとすべてのサブタスクの異常終止をひき起してしまう．なお，DETACH マクロ命令の一般形式は**表 8-6** の通りである．

DETACH マクロ命令が実行されると，制御権は DETACH SVC ルーチン

表 8-6　DETACH マクロ命令の一般形式

〔記号〕	DETACH	tcb のロケーション・アドレス	[, STAE={YES / NO}]

tcb のロケーション・アドレス：除去すべきサブタスクの TCB アドレスが貯えられているフルワードの記憶域のアドレス

STAE：
- YES：サブタスクが異常終止のさい，STAE マクロ命令で指示した出口ルーチンに制御権を渡すことを表示．
- NO：サブタスクが異常終止しても，STAE マクロ命令で指示した出口ルーチンに制御権を渡さないことを表示．

に渡る．DETACH ルーチンはまずつぎの検査を行ない，エラーが発見されると，DETACH マクロ命令を出したタスクを異常終止させる：

- DETACH ルーチンに受渡されたサブタスク TCB のアドレスの妥当性検査．
- 消去しようとしているサブタスクが DETACH マクロ命令を出したタスクのサブタスクであるかどうかの検査．

このあと，サブタスク TCB をサブタスク待行列から取り除き，サブタスクが利用していた主記憶域を解放することになる．というのは，サブタスクが終了しても，ETXR ルーチンではサブタスクの TCB やレジスター保存域などのスペースを解放することは行なわないので，これらのスペースの解放は DETACH ルーチンで行なわれることになる．この場合，消去しようとしているサブタスクがすでに終了しているかどうかによって処理の仕方が異なる．サブタスクがすでに終了しているかどうかは，サブタスク TCB の TCBFLGS フィールドの**完了標識**（completion indicator）で確認される．このビットはタスク終了ルーチンまたは異常終止ルーチンでセットされるものである．

サブタスクがすでに終了していることが完了標識で確められると，まずサブタスクの TCB を親タスクが管理しているサブタスク待行列から取り除く．そのあとサブタスク TCB が占めていたスペース（サブプール 253）とレジスター保存域のスペース（サブプール 250）を解放する．

一方，サブタスクがまだ終了していないことが確かめられると，異常終止ルーチンの助けを借りてそのサブタスクを異常終止させてから，サブタスクの消去を行なう．このために DETACH SVC ルーチンは4バイトの ECB のスペースを確保し，そのアドレスを TCB の中の TCBECB フィールドに挿入する．サブタスクが ATTACH された時に ECB が指定されていた場合には，その ECB アドレスは SVRB に保存される．また，TCB 中の TCBIQE フィールドをゼロにセットして ETXR ルーチンが実行されないようにする．そのあと，DETACH ルーチンは TCBECB でポイントされている ECB で一時待ち状態（wait）に入る．サブタスクが異常終止されるとその旨が ECB に知らされ

るので，DETACH ルーチンは再ぶ活動状態に入る．ここで，サブタスク TCB をサブタスク待行列から除去し，スペースを解放することによって DETACH 機能を完了する．

2. TCB に関する間接サービス機能

TCB に関する間接サービス機能は，大別してつぎの5種類から構成される：

- プログラム割込みエグジット・ルーチンの指定
- 1つ以上の事象とプログラム実行の同期化
- 資源利用の逐次化
- 非同期的エグジット・ルーチンのスケジューリング
- タスクの非同期的エグジット・ルーチンの指定

ユーザーの処理プログラムでプログラム割込みエグジット・ルーチンを指定し，そのルーチンでユーザー・タスクを遂行するプログラムの中で生じたプログラム割込みを処理することができる．監視プログラムではユーザーのエグジット・ルーチンの存在をテストすることができるようになっている．このエグジット・ルーチンの面倒をみる SPIE SVC ルーチンがユーザー・エグジット・ルーチンへの間接的なポインター（後述する PIE のアドレス）を SPIE マクロ命令を出したプログラムの TCB へ挿入する．プログラム割込みが生じたあと，第一次プログラム割込み処理ルーチンが PIE フィールドのアドレスを拠点にユーザー・ルーチンに制御権を受渡し，ユーザー・ルーチンでそのプログラム割込みを処理する．別の場合には，第一次割込み処理ルーチンが異常終止（ABTERM）ルーチンを用いて，エラーで割込みを引きおこしたタスクの異常終止をスケジュールすることになる．

ユーザーのプログラムまたはシステム・プログラムで WAIT と POST マクロ命令を用いて，入出力オペレーションの完了といった**事象**（event）とプログラムの実行とを同期化することができる．WAIT ルーチンは，指定した事象が生じるまで要求プログラムの実行を中断する．その事象が発生すると，POST

ルーチンが**事象制御ブロック**（Event Control Block；略して**ECB**）の中の特定ビットを変えることによって，事象の発生を表示する．その結果，待ち状態にある要求プログラムが作動可能状態になるので，ディスパッチャーはそのプログラムの実行をスケジュールする．これが，特定事象とプログラムとの同期化の基本である．

資源利用の逐次化の機能は ENQ と DEQ ルーチンが受け持つ．これによって，特定資源ないしは特定資源群の利用要求が異ったタスクから出されても，同時にはアクセスできないようになる．それらの要求は順番に処理される．ここでいう資源には，データ・セット，データ・セット内のレコード，プログラムあるいは主記憶装置内の作業域などが含まれる．資源が利用可能な状態にあれば，それを要求したプログラムに制御権が戻される．このとき，資源の利用可能状況を示す戻りコードを一緒に戻すこともできる．一方，要求資源が利用不能だと，つぎのいずれかの方法がとられる．資源が利用可能な状態になるまで要求プログラムが待ち状態に置かれるか，または資源が利用できない旨の戻りコードをつけて要求プログラムに制御権を戻す．その資源をもはや必要としないときには，DEQ マクロ命令を出してシステムに知らせる．DEQ ルーチンでは，待ち状態にある要求プログラムの待ちカウントを減じたうえでその作動可能状態をテストする．資源要求プログラムが作動可能だと，DEQ マクロ命令を出したルーチンに代わってそのプログラムを実行できるかどうか決定してくれる．

タスクの終了状態のように事前に予測がつけがたい事象の特別な処理を行なうのが**非同期的エグジット・ルーチン**（asynchronous exit routine）である．このルーチンのスケジューリングは監視プログラムが行なう．事象が実際に生じたときにそのスケジューリングが開始される．

非同期的エグジット・ルーチンは，タスクの異常終止処理がスケジュールされたときに制御権を受けとるように指示することもできる．このために使用するのが STAE マクロ命令である．このマクロ命令を与ると，異常終止の処理をそのタスクで行なうための前準備をやってくれる．その結果，タスクが異常終止の処理に入ると，ABEND/STAE インタフェース・ルーチンが呼び出され

ユーザー作成の STAE エグジット・ルーチンがスケジュールされる．STAE エグジット・ルーチンで再試行ルーチンをスケジュールするよう表示していると，ABEND/STAE インタフェース・ルーチンは STAE 再試行ルーチンのアドレスのセットも行なう．そのあとで制御権をディスパッチャーに渡す仕組みになっている．

以下で，TCB に関する間接サービスのそれぞれの機能について具体的な説明を加えてゆこう．

1) プログラム割込みエグジット・ルーチンの指定機能

プログラム割込みを起したタスクは，通常はシステムが自動的に異常終止（ABEND）させてしまう．ところが，SPIE マクロ命令を使用すれば，特定タイプのプログラム割込みをユーザー指定のエグジット・ルーチンで処理することが可能となる．

SPIE マクロ命令のオペランドでは，プログラム割込みを処理するユーザー・ルーチンのアドレスとそのルーチンで処理したい割込みの種類を指定することができる（**表 8-7** 参照）．また，SPIE マクロ命令を与えることによって，プログラム割込みを処理するための重要な制御ブロック（PICA および PIE）が作られる．後述するように，SPIE SVC ルーチンはこれらの制御ブロックの連鎖関係を作りあげ，実際にプログラム割込みが生じたときにそれをユーザーのエグジット・ルーチンで処理できるような準備をする．

プログラム割込みを処理するための重要な制御ブロックの１つが**プログラム割込み制御域**（Program Interruption Control Area；略して **PICA**）である．PICA は SPIE マクロ命令がアセンブルされるさいのマクロ展開の結果作り出されるもので，ユーザーのエグジット・ルーチンのアドレスや取り扱うプログラム割込みのタイプに関する情報などが収容される．PICA の形式は**図 8-34** のごとくである．

別の制御ブロックは**プログラム割込み要素**（Program Interruption Element；略して **PIE**）と呼ばれる．PIE はタスクが最初の SPIE マクロ命令を出したときに，サブプール０に作られる．その後 SPIE マクロ命令が出されても PIE

表 8-7　SPIE マクロ命令の一般形式

〔記号〕	SPIE	〔割込みエグジットのアドレス，(割込み)〕

割込みエグジットのアドレス：“割込み”オペランドで指示したタイプのプログラム割込みが生じたとき制御権を受渡すべきエグジット・ルーチンのアドレス

割込み：下記に示す割込みタイプをコードで指定．割込みタイプの指定にはつぎの2種類がある：

- 割込みタイプを示すコードをカンマで区切って列挙する方法．
  (3, 6, 7, 9, 10)
- 2つのコードのペアで割込みタイプの範囲を示す方法．
  ((2, 8),　(10, 13))

コード	プログラム割込みのタイプ	コード	プログラム割込みのタイプ
1	オペレーション割込み	9	固定小数点除算割込み
2	特権命令違反割込み	10	10進数オーバフロー割込み(M)
3	EXECUTE 違反割込み	11	10進数除算割込み
4	記憶保護違反割込み	12	指数オーバフロー割込み
5	アドレス指定違反割込み	13	指数アンダーフロー割込み(M)
6	指定違反割込み	14	シグニフィカンス割込み(M)
7	データ違反割込み	15	浮動小数点除算割込み
8	固定小数点オーバフロー割込み(M)		

(M)はマスク可能な割込み

の中の“PICA のアドレス”フィールドがそれに応じて更新されるだけで，新しく PIE が作られることはない．つまり，PIE はタスクに対して1個しか作られない．**図 8-35** に示したのが PIE の形式である．

さて，SPIE マクロ命令が実行されると SPIE SVC ルーチンが働くのであるが，それがどのように機能するかみておこう．

SPIE SVC ルーチンが行なう処理は，すでに PIE が作られているかどうかによって異なる．換言すれば，同一タスクに対して前に SPIE マクロ命令が実

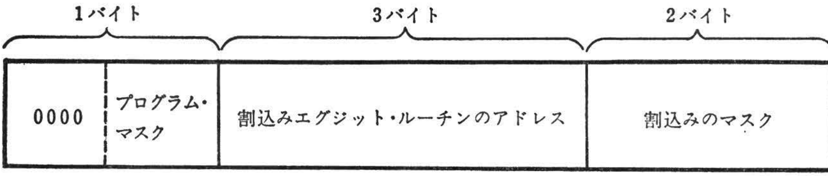


図 8-34 プログラム割込み制御域 (PICA) の形式

行されたことがあるかどうかで処理方法に差がある.

まず, 該当 TCB の中の TCBPIE フィールドをチェックする. TCBPIE フィールドが0の場合には PIE を作りそのアドレスを TCBPIE に入れる (図 8-36 参照). このフィールドが0でない場合には, すでに別の SPIE マクロ命令の実行で PIE が作られていることを意味する. この場合には, PIE の "PICA のアドレス" を汎用レジスター1に入れて, ユーザー・プログラムに返す (なお, PIE が新しく作られた場合はレジスター1は0である). そのあとで, 新しい PICA のアドレスを PIE の "PICA のアドレス" に入れ, 旧アドレスと置きかえる.

さらに, SPIE SVC ルーチンは PICA の中の "プログラム・マスク" フィー

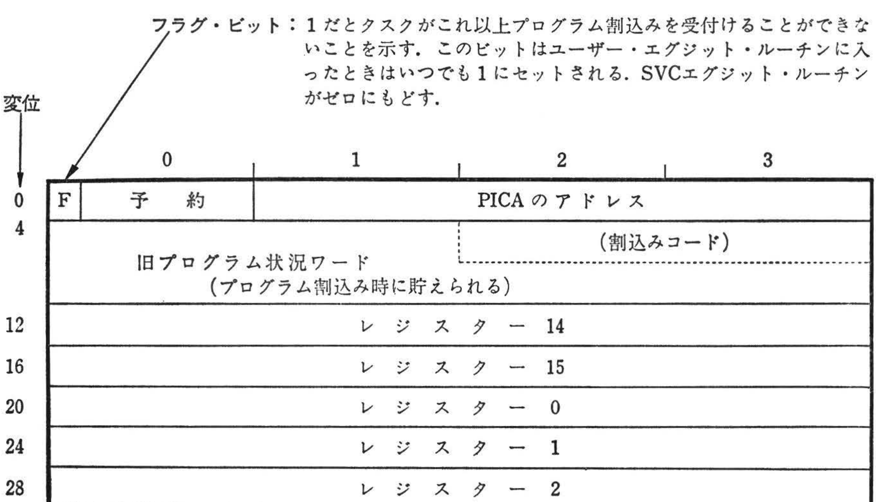


図 8-35 プログラム割込み要素 (PIE) の形式

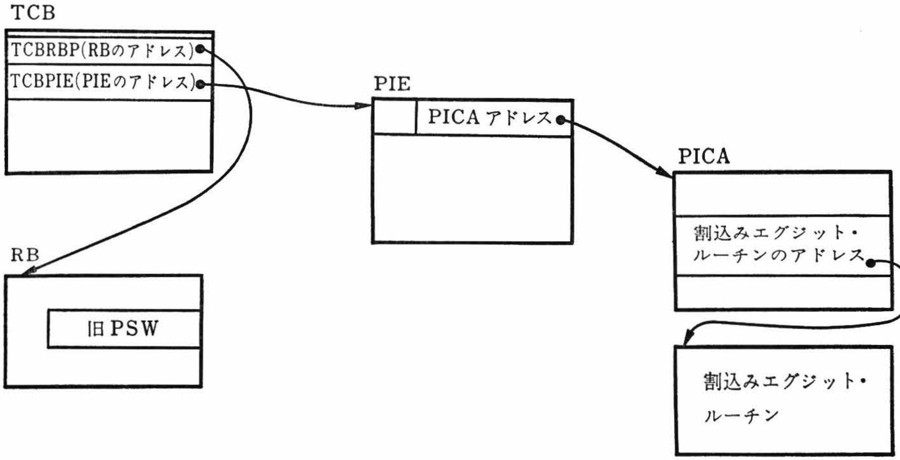


図 8-36　TCB から割込みエグジット・ルーチンへの連鎖関係

ルドの内容を SPIE マクロ命令を出したプログラムに対する要求ブロック（RB）の旧 PSW の“プログラム・マスク”フィールドに入れる．これによって，SPIE のユーザー・エグジット・ルーチンが扱うタイプのプログラムに割込みがマスクされないようにする．このあと，SPIE マクロ命令を出したプログラムに制御権を戻すのである．

SPIE マクロ命令の実行によって，特定プログラム割込みが生じた場合にユーザーのエグジット・ルーチンで処理するための前準備の体制がととのう．実際には図 8-36 に示したような制御ブロック間での連鎖関係が確立し，初期値化が行なわれたことになる．したがって，プログラム割込みが生じたときのユーザー・エグジット・ルーチンのスケジュールは，TCB から制御ブロックの連鎖関係をたどってゆくことによって達成される．このスケジュールの仕方とエグジット・ルーチンの実行について以下で説明する．

プログラムの実行中にプログラム割込みが生じると，監視プログラムの中の**プログラム割込み処理ルーチン**（Program Interruption Handler）に制御権が渡る．これ以降の処理手順を図示したのが，**図 8-37** である．

処理プログラムでプログラム割込みが生じたことが判明すると，プログラム

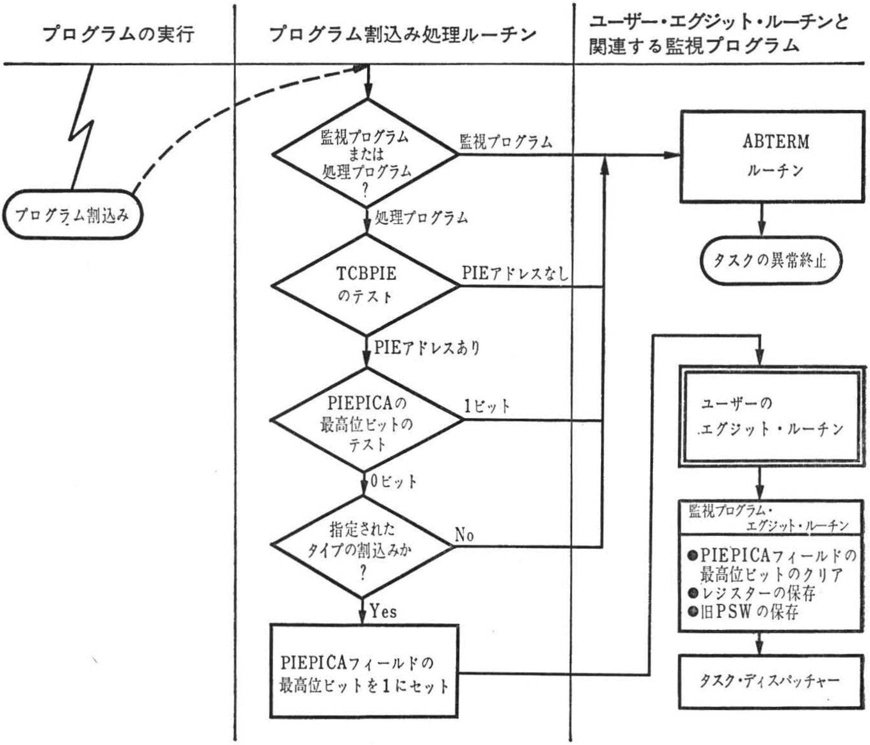


図 8-37 プログラム割込みの処理手順

割込み処理ルーチンは TCB の中の TCBPIE フィールドをテストし，PIE の存在の有無をチェックする．TCBPIE フィールドが0であると，ABTERM ルーチンがとられる．TCBPIE フィールドに PIE アドレスがあるときは，PIE の中の“PICA のアドレス”フィールド（**PIEPICA フィールド**という）の最初のビットが立っているかどうかテストする．このビットが1であることは，ユーザーのエグジット・ルーチンそのものの実行中にプログラム割込みが生じたことを意味する．この場合にはやむを得ずタスクを異常終止させることになる．

PIEPICA の最高位ビットが0であると，割込みの種類が SPIE マクロ命令のオペランドで要求されたものかどうか検査する．このために，旧 PSW の割込みコードと PICA の中の“割込みのマスク”フィールドを調べる．指定さ

れた割込みタイプに該当すると，ユーザーのエグジット・ルーチンをスケジュールすることになる．ユーザー・エグジット・ルーチンをスケジュールする時には PIEPICA の最高位ビットを1にし，ユーザー・エグジット・ルーチンの実行中にプログラム割込みが生じた場合の識別ができるようにする．そのあとで，エグジット・ルーチンに制御権を移す．

ユーザーの割込みエグジット・ルーチンに制御権が渡ったときの汎用レジスターの内容はつぎの通りである：

- レジスター0：制御プログラムの情報．
- レジスター1：割込みを引き起したタスクの PIE のアドレス．
- レジスター2—12：プログラム割込みが生じた時の内容と同じ．
- レジスター13：主プログラムの保存域のアドレス・エグジット・ルーチンではこの保存域を用いてはならぬ．
- レジスター14：制御プログラムへの戻りアドレス．
- レジスター15：エグジット・ルーチンのアドレス．

ユーザーのエグジット・ルーチンの中で PIE の内容，特に旧 PSW のつぎの**命令アドレス**（next instruction address）を変更するのは自由である．

エグジット・ルーチンの最後で RETURN マクロ命令が実行されると，プログラム割込み処理ルーチンの直前にある SVC 命令に戻る．この命令が実行されて SVC 割込みが起ると，SVC の旧 PSW とプログラム割込みの新 PSW のつぎの命令アドレスの内容が比較される．両者の内容が等しいことでユーザー・エグジット・ルーチンから戻ったことが判明する．その結果，PIE の中の PIEPICA フィールドの最高位のビットを0に戻す．ここで，監視プログラムのエグジット・ルーチンはレジスターの内容を該当 TCB の所定フィールドに保存する．また，RB の旧 PSW フィールドに PSW を保存する．この場合の PSW は，その左半分は SVC 旧 PSW から，右半分は PIE の旧 PSW から構成される．これによって，処理プログラムへの戻り点をかえ，PIE が破壊されている危険性を避けることができる．

監視プログラム・エグジット・ルーチンで TCB にレジスターを保存し，RB に

旧PSWを作りあげてから,タスク・ディスパッチャーに制御権を渡すのである.

2) 特定事象とプログラム実行の同期化機能

事象が起きるのを待つことおよび事象が起ったことを知らせるのにはWAITとPOSTとの機能が使われる. WAIT, POSTのメカニズムは監視プログラムの一部であるWAIT SVCルーチンとPOST SVCルーチン, および制御ブロックであるRBとECBとで構成される.

WAITルーチンは, 待つべき事象を表示するECBに“待ち”ビットを立て, WAITマクロ命令を出したプログラムに対応するRBに完了を待っている事象の数(待ちカウント)を入れる(図8-38のRBWCFフィールド参照). RBに待ちカウントが入っているプログラムはCPUを利用することができないことを意味する. つまり, タスク・ディスパッチャーはRBのRBWCFフィールドをチェックしてそのプログラムのディスパッチの可否の判断をする.

一方, POSTルーチンは, 事象が生じたことを知らせるためにECBの待ち標識をゼロにし, 逆に完了標識に1ビットを立てる. さらに, その事象が生じるのを待っているプログラムに対応するRBのRBWCFフィールドから1を引く. RBのRBWCFフィールドが0になると, 待つべき事象のすべてが起

<table>
<tr><td colspan="3">0
(予　　約)</td></tr>
<tr><td colspan="3">4
RBABOPSW：ABTERMルーチン実行後のユーザーの旧PSWの右半分. そうでない場合は0が入る</td></tr>
<tr><td>8 RBWCSA：待ちカウントの保存域</td><td>9
RBSIZE：このRBのサイズ</td><td>10
RBSTAB：状況ビットおよび属性ビット</td></tr>
<tr><td>12
RBCDFLGS：内容制御標識</td><td colspan="2">13 RBCDE：この要求ブロックが関連するモジュールに対応する内容登録簿の記入項目(entry)のアドレス</td></tr>
<tr><td colspan="3">16
RBOPSW：ユーザーの旧PSW</td></tr>
<tr><td>24
常にゼロ</td><td colspan="2">25 RBPGMQ：同一逐次再使用プログラムの使用要求を表示している要求ブロックのアドレス</td></tr>
<tr><td>28 RBWCF：完了を待っている事象の数(待ちカウント)</td><td colspan="2">29 RBLINK：RB待行列中の直前の要求ブロックのアドレス, ないしはこの要求ブロックがRB待行列の最初にある場合だとTCBのアドレス</td></tr>
</table>

図 8-38 プログラム要求ブロックの形式

表 8-8　WAIT マクロ命令の一般形式

〔記号〕	WAIT	〔事象の個数，〕{ECB＝アドレス / ECBLIST＝アドレス}
		事象の個数：最大 255 の数．このオペランドを省略すると 1 とみなされる．事象の個数は事象制御ブロック（ECB）の数を越えてはならない． ECB：処理の続行のまえに生じる必要のある 1 個の事象を表わすECBのアドレス．事象の個数が 1 と指定されているか省略されているときに有効． ECBLIST：ワード境界に合わせて確保された連続したフルワードの記憶域アドレス．各々のフルワードには ECB のアドレスが収容されていること．最後のフルワードの最高位ビットはリストの最後を表示するために 1 にセットされている必要がある．

ったことを意味しその RB に対応したプログラムは作動可能状態となる．

①　WAIT SVC ルーチンの機能

プログラムで WAIT マクロ命令を出すと WAIT ルーチンに制御権が渡る．WAIT マクロ命令では，そのオペランドで ECB アドレスまたは ECBLIST アドレスを使って待つべき事象に対応する ECB を指定する．さらに，これらの事象のうちのいくつが起ると待ち状態から解放されるかを指示する．このマクロ命令の一般形式は表 8-8 で示している．WAIT マクロ命令で指定した待つべき事象の個数が ECB の個数より少ない場合，つまり事象のうちの一定個数だけが起これば待ち状態から抜けることができる時には，WAIT SVC ルーチンがマクロ命令を出したプログラムに対応する RB の RBSTAB フィールドの中に“検索”標識を立てる．この“検索”標識は POST ルーチンでチェックする．“検索”標識は待ち状態を抜けても，まだ POST されていない ECB が存在することを意味する．しかも，それらの ECB は待つ必要はなくなっている．したがって，POST SVC ルーチンが RB の中の待ちカウント（RBWCF

フィールド）を0にしたときに“検索”標識をチェックし標識が立っている場合には，まだ待ち標識が立っているECBを探し出し，その標識をリセットしてくれるのである．

逆にWAITマクロ命令で指定された待つべき事象の個数がECBの個数よりも多い場合だと，マクロ命令を出したタスクを異常終止させる．また，WAITマクロ命令を出したプログラムの記憶保護キーが0でないとき，つまり問題プログラム状態である場合には，指定されたECBアドレスについてつぎの3種類の検査を行なってくれる：

- フルワード境界にあるか
- 該当システムの主記憶装置のアドレス内にあるか
- ECBの保護記憶がWAITマクロ命令を出したタスクのTCBにある記憶保護キー（TCBPKFフィールド）と合致するか

これらECBアドレスのチェックに違反したタスクは異常終止されることになる．ただし，WAITマクロ命令を出したプログラムが記憶保護キー0のもとで働いている場合だと上記の検査は行なわれない．

ECBアドレスの検査が終ると，WAIT SVCルーチンはECB内の待ち標識と完了標識とをつぎの手順で調べて所定の処理を行なう（ECBの形式については**図8-39**を参照のこと）：

i）すでに待ち標識に1ビットが立っている時はタスクを異常終止させる．

ii）待ち標識のビットが立っていないときは，完了標識を調べそれが1ビットにセットされていると，待ちカウントから1を引く．ECBの内容はそのままの状態にしておく．

iii）待ち標識のビットも完了標識のビットも立っていない場合には，ECBの待ち標識を1ビットにセットし，さらにECBの完了コードのフィールドにWAITマクロ命令を出したプログラムのRBのアドレスを挿入する．このRBアドレスは，あとでPOSTルーチンがRBの中の待ちカウントをマイナスするのに用いる．

ビット 0	1	2 〜 31
待ち標識(W)	完了標識(C)	完了コードまたは RB アドレス

標識の状態　　　　**標識の意味とビット2〜31の内容**

W　C

0　0………事象が起きるのを待つこともなかったし，事象が完了したことを知らされもしなかった．ビット2〜31のの内容は無意味な情報である．

0　1………事象が完了したことが知らされ，もはや待ち状態ではない．ビット2〜31の上位ビットに完了コードが収容されている．下位ビットはゼロである．

1　0………事象が起きるのを待っているが，まだ完了したことは知らされていない．ビット2〜7はゼロで，ビット8〜31に WAIT マクロ命令が出された RB のアドレスが入っている．

1　1………この状態は普通起り得ない．

完了コード　　　　**意　味**

111111………正常に完了（エラーなし）

000001………入出力エラー・コード

000010………エクステント・エラー・コード．これは，IOB で指示されたシーク・アドレスが DEB で指示されているアドレス範囲を越えていることを意味している．

000100………IOB インターセプト・コード

001000………このコードは，入出力要求が開始されることもなかったし，除去されることもなかったことを示す．

001111………エラー再試行ができないことを示す．このコードはエラー回復手順中にホーム・アドレスも R0 も読めなかったことを示す．

図 8-39　事象制御ブロック（ECB）の内容

なお，**ii**）の処理結果待ちカウントが0になったときは，すでに待ち状態にある必要がなくなったことを意味するので，WAIT マクロ命令で受渡されたすべての ECB の待ち標識をゼロにリセットする．

WAIT マクロ命令のパラメータとして引き渡されたすべての ECB について上記の処理を行なったあとで，待ちカウントを RB の RBWCF フィールドに

挿入する. 待ちカウントが0でないことは WAIT マクロ命令を出したタスクが WAIT することを意味するので, IEATCBP の新 TCB ポインターを0にしてこの旨をエグジット・ルーチンに知らせる.

WAIT SVC ルーチンは, さらに **“待ち状態制限** (wait limit)” を行なうかどうかをチェックする. “待ち状態制限” は, ジョブ・ステップが起こり得ない事象を待って無駄な待ち状態に入ってしまうのを防ぐもので, つぎの条件が満されたときだけに行なう:

i) 開始プログラムの TCB の TCBTME フィールドが**タイマー待行列要素** (Timer Queue Element ; 略して **TQE**) を指示しており, ジョブ・ステップ・タイミングが行なわれている.

ii) ジョブ・ステップ内のすべてのタスクが WAIT マクロ命令により待ち状態にある.

この2つの状態が満足されると, WAIT SVC ルーチンはつぎの処理を遂行することによって, “待ち状態制限” を行なう:

- 開始プログラムの TQE をマイマー待行列から除去する.
- CPU タイムの残り時間を TQE の中の特定フィールドに保存する.
- 30分間またはSMFで指定した “待ち状態制限” を TQE に入れ, “待ち状態制限” TQE に変換する.
- この TQE をマイター待行列に挿入する.

以上の処理を行なってから, WAIT SVC ルーチンは**タイプⅠエクジット・ルーチン** (type I exit routine) にブランチするのである.

② POST SVC ルーチンの機能

事象が生じたことは, POST SVC ルーチンによって ECB と RB とに記録される. この POST SVC ルーチンへは POST マクロ命令と入出力監視プログラムをはじめとしたシステム・ルーチンからのブランチによって制御権が移される. なお, POST マクロ命令の一般形式は**表 8-9** で示した通りである.

表 8-9　POST マクロ命令の一般形式

〔記号〕	POST	ecb のアドレス〔，完了コード〕

ecb のアドレス：事象を表わす事象制御ブロック(ECB)のアドレス

完了コード：0から $2^{24}-1$ の値．この完了コードが指定されていないと0とみなされる．

WAIT SVC ルーチンの場合と同様，POST SVC ルーチンでも問題プログラム（記憶保護キーが0でない場合）で POST マクロ命令を出したときは POST すべき ECB のアドレスのチェックを行なう．チェックの内容は WAIT SVC ルーチンの場合と全く同じであり，いずれかに違反していると POST マクロ命令を出したタスクは異常終止させられる．

つぎに，ECB の待ち標識にビットが立っているかどうかチェックする．ビットが立っているときには，ECB の完了コードのフィールドに RB アドレスを持っているので，このアドレスを参照してつぎの検査を行なう：

- RB アドレスが該当システムの主記憶装置内のフルワード境界にあるか．
- RB の中に収容されている旧 PSW のシステム・マスクが全部1であるか（WAIT SVC ルーチンでシステム・マスクはすべて1にされている）．
- ECB でポイントしている RB の旧 PSW の記憶保護キーと待ち状態にあるプログラムの RB の旧 PSW の記憶保護キーとが一致するか．
- 待ち状態にあるプログラムで最後に実行した命令は WAIT マクロ命令であるか．

これらのいずれかに当てはまらない場合には，POST マクロ命令を出したタスクは異常終止させられる．

つぎには，ECB の完了標識のチェックに移る．完了標識にあらかじめ1ビットが立っていると，事象はすでに完了していることを意味するので，POST SVC ルーチンのこれ以降の処理はスキップされる．逆に，完了標識にビット

が立っていないと，つぎに示す処理に入る：

i) 待ち標識ビットが立っていると，それを0にする.

ii) 完了標識のビットを1にし，完了コードを ECB に入れる.

iii) 待ち状態にあるプログラムの RB の中の待ちカウントから1をひく．ただし待ちカウントがすでに0であると1を引くことはない．これはタスクがすでに ABTERM ルーチンによって異常終止させられた場合である.

iii) の処理の結果，待ちカウントがまだ0にならないときは POST SVC ルーチンの処理はこれで終了する．一方，待ちカウントが0になると，さらにつぎの処理に移る：

- 待ち状態にあるプログラムに対応する RB の中の RBSTAB フィールドに "検索" 標識のビットが立っているときは，ECBLIST で指定してあるすべての ECB の待ち標識を0にリセットする.
- 開始プログラムの TCB の中にある TCBTME フィールドが TQE をポイントしているかどうか調べる．TQE があると，それが REAL タイプか TASK タイプかテストする．REAL タイプだとジョブ・ステップ全体が待ち状態にあり，"待ち状態制限" がとられていたことを意味する．一方，RB の待ちカウントが0になったことは待ち状態が解けたのであるから，WAIT SVC ルーチンが TQE に一時保存しておいた CPU 時間を元に戻す．さらに，TQE を TASK タイプにして TQE をタイマー待行列に入れる.

この様な処理を終ったあとで，**タスク切換えルーチン** (task switching routine) に移り，待ちカウントが0になったプログラムのディスパッチができるかどうかをつぎの様にしてチェックする：

- 待ちカウントが0になった RB がそのタスクの RB 待行列のトップにあるか.
- TCB の中に "ディスパッチング不能" ビットが立っていないか.
- 作動可能状態になったタスクの指名順位が目下 CPU を使用しているタス

クのそれより高いか.

これら3つの条件が満されると，IEATCBP の新 TCB ポインターに作動可能状態になったタスクの TCB アドレスを挿入する.

以上が POST SVC ルーチンの機能である. なお，POST マクロ命令によって POST SVC ルーチンに入った場合だと，タイプⅠエグジット・ルーチンを経由してエグジットする. 一方，システム・ルーチンからブランチして POST SVC ルーチンに入った場合にはブランチによって元のルーチンにもどる仕組みとなっている. したがって，後者の場合にはシステム・ルーチンからタイプⅠエグジット・ルーチンないしはディスパッチャへ行ったときはじめてタスクの切換えが起こる.

3）システム資源の逐次使用化の機能

逐次使用を行なう必要のあるシステム資源に対してその利用要求が出されると，それらの要求は順番に処理する必要がある. システム資源の逐次使用は ENQ/DEQ マクロ命令を出すことによって可能になる. 逐次使用の必要がある資源に対して名前をつけ，その名前を使って ENQ/DEQ 命令を出す. その結果，資源とそれに対する要求を表わす一連の制御ブロックができる. これらの制御ブロックと RB の待ちカウントを使ってシステム資源の逐次利用の制御を行なうのである. 制御の仕組みを明らかにする前にまず逐次再使用可能な資源の使い方と ENQ/DEQ マクロ命令が果たす機能を概観することにしよう. そのあとで，ENQ/DEQ の制御の仕組みと SVC ルーチンが行なう機能を解明してゆこう.

①　逐次再使用可能資源の利用

逐次再使用可能（serially reusable）資源の代表例には，逐次再使用可能な属性を指定したロード・モジュールがある. これに関しては第4章の第3節「プログラム構造と制御プログラム」の個所でふれた. 逐次再使用可能なロード・モジュールの利用にあたっては，そのロード・モジュールの同時使用を回避することの必要性を強調した. このことは，複数個のユーザーがシステム資源を

修正するようなときの逐次再使用資源のいずれにも当てはまることである.

たとえば，あるジョブ・ステップで複数個のタスクに関連するプログラムが使用しているデータ域を考えてみよう. このデータ域もシステム資源である. ユーザーのいくつかでそのデータ域のレコードの読み取りだけしか行なっていないとしよう. レコードの変更は一切行なっていないので，該当データ域を同時に複数個のユーザーが利用しても何ら問題は生じない. 逆に，あるユーザーがそのデータ域のレコードを読みとり，更新し，もとに書き戻す操作をしていたらどうであろう. それぞれのユーザーは1時点で同時にレコードを取り出したり，更新したり，書き変えを行なったりすることは許されない. さらに，あるユーザーが更新しているレコードの読み取りだけを希望しても，そのレコードの書きかえが完全に終了するまで待たざるを得ない. 逐次再使用可能資源はいずれもこのような方法で利用せざるを得ないのである.

単一タスクの遂行中に使用する逐次再使用可能資源に関しては，その資源の誤った利用をしないよう自分で気をつける必要がある. つまり，最初の使用が完了するまでつぎの使用をしないようなプログラム論理を組み立てることだ. とりわけ，逐次再使用可能資源をエグジット・ルーチンで利用するときに注意を要する. エグジット・ルーチンにはプログラム論理とは関係なく非同期的に制御権が渡ってしまうので，ちょうど主プログラムで使用中の資源が利用される恐れがあるからだ. ところで，2つ以上のタスクで必要とする逐次再使用可能資源の利用に関しては，逐次利用を確実に行なえるようにするために ENQ マクロ命令が用意されている. この命令は，単一タスク内の資源の同時使用を回避するのには適用できない.

ENQ マクロ命令は，活動状態のタスクに資源の制御権を割り振るように制御プログラムに要求する. 制御プログラムは資源の利用状況を判定し，活動状態のタスクに制御権を戻すことによって要求を満すか，活動中のタスクを待ち状態にして制御権の割り当てを一時延期するかの方法をとる. 待ち状態のタスクに制御権を移すことができるような状況に資源の利用状況が変化すると，該当タスクは待ち状態を抜け出し，作動可能状態におかれる. この ENQ マクロ命令の一般形式は**表 8-10** に示している. また，ENQ マクロ命令のパラメータ

表8-10　ENQ マクロ命令の一般形式

[記号名]	ENQ	(qname アドレス, rname アドレス, {E / S}, (rname の長さ), {SYSTEM / STEP}, ………) [, RET=TEST / , RET=USE / , RET=HAVE / , RET=CHNG]

の指定内容は表8-11で説明している．この2つの表を参照しながら以下でENQ マクロ命令の機能を言及してゆこう．

② システム資源の名前

要求するシステム資源の表現には，qname と rname と呼ばれる2種類の名前を使用する（**表8-10**参照）．この名前は，資源の実際の名前と関連づけたものであっても，全く無関係のものであっても一向に構わない．制御プログラムは，名前と実際の資源を関係づけることはしない．単に**先入-先出し法**(FIFO)にもとづいて同一の qname と rname がつけられた要求を処理するだけである．名前と実際の資源を関連づける責任はプログラマーにある．同一資源を表示するために qname と rname を使用するのは，その資源を使用する人すべての責任であるといえる．制御プログラムは，異った qname と rname の組合せだと異った資源として取りあつかうからである．

特定のジョブ・ステップ内のタスクの遂行だけに使用される性格のシステム資源だと，適当な qname と rname の組合せでよい．この場合には，ENQ マクロ命令の STEP オペランドを指定する．このパラメータは要求資源がそのジョブ・ステップ内だけで使用されることを意味する．この指定をすると制御プログラムは rname にジョブ・ステップ識別名を附加する．その結果，無意識のうちに別のジョブ・ステップで重複した qname と rname とが使われるといった事態は起らない．ところが，システム内のどのジョブ・ステップでも利用可能なシステム資源だと，qname と rname は全ての利用者が合意した名前でないと困る．同一の qname と rname の組合せを用いることによって同

一資源を表示するのであるから，名前と資源の対応づけをした一覧表を作って管理する必要がある．ともあれ，この種の資源を要求する場合には，ENQ マクロ命令のオペランドで SYSTEM と指定する（**表 8-11**）．

③ “排他的”要求と“共用”要求

ENQ マクロ命令でＥまたはＳのいずれかを指定することによって，あるタスクに対する資源の排他的制御か共用制御かを要求することができる．資源の利用がその修正と更新を伴なう場合には，排他的制御を要求しなければならない．たとえば，逐次再使用可能ロード・モジュールを利用することを要求し制御権を渡す場合だと，プログラムの実行中にその一部を修正するので排他的制御の要求が必要となる．一方，レコードを読み取るだけでその変更を一切伴なわないとしたら，共用制御の要求ができる．

逐次再使用資源のすべての利用者を保護するためには，上記の基準にもとづいて排他的制御か共用制御かの要求をなすべきである．ともあれ，排他的要求に応じて資源の制御権があるタスクに与えられると，その資源の制御権は他の一切のタスクには渡されない(その資源の利用が完了するまでは)．逆に，共用要求に応じて資源の制御権があるタスクに渡された場合だと，別の共用制御の要求に応じて同時に他のタスクにも制御権が与えられる．ただし，別の要求が排他的制御であれば制御権は与えられない．上記の規則にしたがっているかぎり，共用制御の要求は他のタスクで行なう資源の修正・更新に対して完全に保護されるのである．

④ 要求の処理の仕方

制御プログラムは，qname と rname との組合わせのそれぞれに対するリストを作りあげる．さらに，ENQ マクロ命令が出されたときに活動状態にあったタスクを表示するための記入項目をリストに入れる（後述する QCB, QEL を参照のこと)．すでにリストが存在している qname と rname の組合せを指定した要求を受けとったときは，その記入項目だけを現存するリストの中に作りあげる．qname と rname の組合せに対応したリストがまだ存在していない場合だと，新しいリストを作りあげてから，記入項目を入れる．タスクを表示

表8-11　ENQ マクロ命令のパラメーター一覧表

パラメータ	指定内容と意味
qname アドレス	8桁の名前を貯えている主記憶域のアドレス．逐次再使用資源の利用を要求するプログラムではすべてその資源を表わすために同じ qname と rname を用いなければならない．
rname アドレス	資源を表わすために qname とともに用いる rname を貯えている主記憶域のアドレス．この名前は修飾することが許され，1桁から 255 桁までの長さを用いる．
E	資源の排他的制御を要求するときに指定．このオペランドが省略されると，排他的制御の要求とみなされる．タスクの制御中にその資源が修正更新される場合には，排他的制御の要求をすべきである．
S	資源の共用制御を要求するときに指定．タスクの制御中にその資源が修正更新されることがないなら，共用制御を要求すべきである．
rname の長さ	上述の rname の長さ．省略すると，アセンブルされた rname の長さが用いられる．アセンブルされた長さを無視するために1から 255 の値を指定することができる．またゼロの値を指定してもよい．この場合には rname の長さは，rname アドレスの最初のバイトに収容されていなければならない．
STEP	この命令を出したプログラムのジョブ・ステップ内だけで資源が用いられ，他のジョブ・ステップのプログラムから同じ qname と rname の要求が行なわれても別の異った資源を意味することを指示する．
SYSTEM	2つ以上のジョブ・ステップのプログラムで資源が用いられ，システム内の他のジョブ・ステップのプログラムで同じqname と rname の要求が行なわれると同じ資源を意味することを指示する．
RET=	ENQ マクロ命令で指示された資源に対する条件付き要求をする．このオペランドが省略されると無条件要求となる． TEST：資源の利用可能状態のテストを要求するが，資源の制御権は要求しない． USE：資源が即時に利用可能であればその制御権を活動状態のタスクに割り振ることを指示．資源がいずれも利用不可能でも，活動状態のタスクは待ち状態にはおかれない． HAVE：資源要求が同じタスクに対して以前になされていないときだけ，資源の制御権を要求することを指示． CHNG：活動状態のタスクで制御されている資源に対してその属性を共用から排他的に変更することを要求．

する記入項目（QEL）は，制御プログラムが要求を受けた順番にリストに入れられる．つまり，この場合にはタスクの優先順位は何ら影響しない．タスクへの資源の制御権の割り振りは，つぎの２つの要素にもとづいて行なわれる：

- タスクを表示している記入項目のリスト上の位置
- 記入項目をリストに追加させた要求が排他的制御であるか，共用制御であるか

制御プログラムはこの２つの条件を考慮に入れて，資源の制御権をタスクに割り当てることができるかどうか決定する．この場合のやり方を**図 8-40** をもとに具体化しておこう．

図 8-40 は極めて一般的な qname と rname に対して作られたリストの状況だけを示している．記入項目（ENTRYn）のつぎのＳとＥとはそれぞれ排他的制御の要求か共有制御の要求かを示している．資源の制御権は必ずリスト上の最初の記入項目で表示されているタスクに渡されるので，図では ENTRY 1 で表示されているタスクに資源が割り振られる．ENTRY 2 を作りあげた要求は排他的制御であるので，それに対応するタスクは，リスト内の他のすべての記入項目で表示されているタスクと同様に待ち状態におかれている．

該当資源の制御権が ENTRY 1 で表示されているタスクから解放されると，その記入項目はリストから除去される．図 8-40 のステップ２で示しているよ

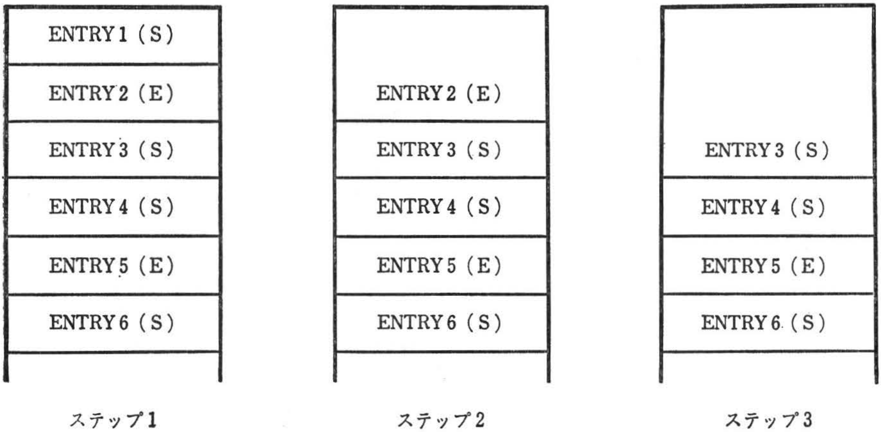


図 8-40　ENQ マクロ命令による処理

うに，ENTRY 2 がリストの最初に位置するようになる．したがって，これに対応するタスクに制御権が割り振られる，ENTRY 2 を作り上げた要求は排他的制御であるので，他の記入項目で示されているタスクはすべて待ち状態のままである．

図8-40のステップ3では，資源の制御権が ENTRY 2 のタスクから解放されたあとのリストの状況を示している．ENTRY 3 がリストの最高位にあるので，つぎは ENTRY 3 で表示されているタスクに資源の制御権が移る．ENTRY 3 は資源が共用できることを示しており，かつ ENTRY 4 も同様であることから ENTRY 4 にも資源の制御権が渡される．この場合，ENTRY 5 は排他的制御となっていることから，これに対応するタスクには，ENTRY3 と ENTRY 4 の双方のタスクから資源の制御権が解放されないかぎり，制御権の委譲は生じない．このようにして，リストの残りの記入項目も処理されてゆく．

以上のことをまとめると，制御プログラムはつぎの一般的規則にもとづいて資源の制御権の割り振りを行なっていることになる：

- リストの中の最初の記入項目で表示されているタスクに必ず資源の制御が与えられる．
- 要求が排他的制御であると，その記入項目がリストの最初の記入項目にならないと，資源の制御権は該当タスクに与えられない．
- 要求が共用制御であると，対応記入項目がリストの最初である場合，またはリスト内のそれより前に位置する記入項目のすべてが共用制御を示している場合のどちらでも制御権が与えられる．
- 複数資源に対する要求の場合には，排他的要求のすべての記入項目がリストの最初にあるとき，および共用要求のすべての記入項目が最初にあるか別の共用要求の記入項目が前にあるときに，該当タスクに制御権が移る．

⑤ END と DEQ の適切な使い方

END と DEQ のマクロ命令を適切に使うためには，重複要求を避けること，システムのインタロックを避けることなどがある．END と DEQ マクロ命令の適切な使用に関して以下でいくつかのポイントを紹介しておく：

重複要求の回避

タスクに該当資源の制御権が割り振られていたり，その資源を待っている状態にあるにもかかわらず，同一資源を要求する ENQ マクロ命令を出したときに重複要求の現象が生じる．２番目の要求がリストの記入項目になると，制御プログラムはその矛盾を認知し，最初の要求に対して該当タスクを作動可能状態にし同時に２番目の要求に対しては待ち状態にすることを拒絶する．その結果，２番目の要求によってタスクは異常終止してしまう．したがって，プログラムを作成するさい，最初の使用から資源の制御権が解放されるまで２番目の要求を出さないようなロジックを検討することである．このことは，特にエグジット・ルーチンで ENQ マクロ命令を使用するさい気をつける必要がある．

資源の制御権の解除

ENQ マクロ命令を使ってタスクに割り振った逐次再使用可能資源を解放するのには，DEQ マクロ命令を使用する（**表 8-12** 参照）．タスクは資源の制御権を持っていなければならない．もし，タスクが制御権を持っていないとその資源の制御権を解放することはできないからだ．すでにみてきたように，あるタスクに資源の制御権が割り振られている間，多くのタスクを待ち状態におくことができる．これは，システムが遂行する仕事量を減少させる恐れがある．したがって，可能なかぎり早目に DEQ マクロ命令を出して資源の制御権を解放し，別のタスクの遂行ができるようにする必要がある．

条件つき要求と無条件要求

ENQ と DEQ マクロ命令の普通の使い方は，無条件要求を出すやり方である．実はこれまでに説明してきたのは無条件要求だけである．すでにふれたように，同一タスクを遂行している同じ資源に対して DEQ マクロ命令を介在させずに２個の ENQ マクロ命令を出すとそのタスクは異常終止させられる．ま

表 8-12 DEQ マクロ命令の一般形式

〔記　号〕	DEQ	(qname アドレス，rname アドレス，〔rname の長さ〕， { STEP { SYSTEM }，………)〔，RET=HAVE〕

た，まだ資源の制御権が割り振られたことのないタスクの中で DEQ マクロ命令を出すとやはり同じことになる．このような異常終止の状態を避けるためには，プログラムのロジックを十分注意して設計するか，これから説明する END と DEQ マクロ命令の RET オペランドを使用することである．この RET オペランドが制御権の条件つき 要求ないしは 制御権解放の 条件つき 要求である．

RET＝TEST と指定すると，対応する qname と rname の組合せに対するリストの状況をテストすることができる．この指定をするとリストに記入項目を作ることはない．その代り，要求時のリストの状況を示すコードを与えてくれる．戻りコード8は，同じタスクの記入項目がすでにリストに存在していることを示す．戻りコード4は，無条件要求であったらそのタスクは待ち状態に置かれたであろうことを示す．戻りコード0は，無条件要求であったらそのタスクに資源の制御権が与えられたであろうことを示す．RET＝TEST は，タスクに資源の制御権が割り 振られたかどうかを 判定するのに 使うと 効果的である．逆に，リストの現況を判断しそれにもとづいてアクションを取るのに使うのにはあまり役立たない．というのは，制御プログラムで状況をチェックし，そしてその戻りコードをユーザー・プログラムでチェックして ENQ マクロ命令を出す間に，別のタスクが活動状態になりリストの状況が変化してしまうからである．

RET＝USE は，該当資源が即座に利用可能なときだけ活動状態のタスクに資源の割り振りを行なうように指示する．戻りコードが0だと，リストにすでに記入項目が存在し，該当タスクに資源の 制御権が 与えられていることを示す．戻りコードが4の場合には，無条件要求であると該当タスクが待ち状態に置かれるであろうことを示す．一方，戻りコード8だと，同一タスクに対する記入項目がすでにリストに存在していることを意味する．RET＝USE は，該当資源を使用しないで 遂行できる 別の 処理があるときに用いると 効果的である．別の処理があるかぎり，該当資源を待つ必要はないからである．

RET＝CHNG は，すでに 待ち行列に入れられている資源に対して排他的制御を行ないたい旨の指示をする．戻りコード0は，その資源が即時利用可能状

態にあり，要求タスクの排他的制御に委ねられたことを示す．戻りコード4は，目下別のタスクとその資源を共用しているので，属性変更要求が認められないことを示す．戻りコードが8だと，属性変更を要求したとき活動中のタスクでその資源を ENQ していなかったことを示す．本来なら，これはエラー状態であるが，制御権はユーザーに戻される．

RET=HAVE は，ENQ と DEQ マクロ命令の双方で用いられる．同じタスクに対する資源要求がこれまでになされていないなら，ENQ マクロ命令は通常の制御要求として処理される．戻りコード8は，すでに同じタスクに対する記入項目がリスト上に存在することを示す．戻りコードが0だと，そのタスクに資源の制御権が割り振られたことを意味する．一方，そのタスクが資源の制御権を持っているなら，DEQ マクロ命令は制御権を戻すための通常の要求として処理される．戻りコード0は，資源の制御権が解除されたことを示す．戻りコード8は，資源の制御権がそのタスクに渡っていないことを意味する．RET=HAVE は異常終止を回避するために，エグジット・ルーチンで用いると効果的である．

インタロックの回避

2つのタスクがお互に相手の完了を待ち，しかもいずれのタスクも処理を完了させるのに必要な資源の利用ができないと，インタロック状態が生じる．いわゆる“にっちもさっちも”いかない状況である．**図 8-41** でこの状態の例を示している．タスクAは資源Mを排他的にアクセスしており，それより優先順位の高いタスクBは資源Nを排他的に要求している．資源MはタスクAだけか

タ ス ク A	タ ス ク B
⋮	
ENQ (M, A, E, 8, SYSTEM)	⋮
⋮	ENQ (N, B, E, 8, SYSTEM)
⋮	⋮
	ENQ (M, A, E, 8, SYSTEM)
ENQ (N, B, E, 8, SYSTEM)	⋮
⋮	

図 8-41 インタロック状態の例

らアクセス可能な状態にあるので，タスクBが資源Mの排他的アクセスを要求すると待ち状態に置かれてしまう．一方，資源NはタスクBからのみアクセスできる状態にあるので，タスクAが資源の排他的使用の要求を出すと完全なインタロック状態になる．双方のタスクが待ち状態におち込み一歩も先に進めない状況になってしまう．タスクAが2番目の資源要求を出す前にタスクBが2つの資源MとNに対してまとめて1つの要求をした場合も同様のインタロックが生じてしまう．しかしながら，双方のタスクで複数資源に対して1つの要求をだす方法をとるならインタロックは発生しない．どれかの資源を要求する一方のタスクも待ち状態におちいるが，この場合にはいずれ待ち状態から抜け出すことができるのであり，インタロックを引き起こす原因とはならない．

上記の例は2つのタスクと2種類の資源とがからみあった場合であり，比較的単純なインタロック状況である．実際には，もっと多くのタスクと資源が関係する場合が多い．したがって，なんとしてもインタロック状況を回避するように配慮することが肝要である．そのために，下記のような方法をとることによってインタロック状態を未然防止する必要がある：

- 即刻必要としない資源を 要求しないこと． 逐次再使用可能資源を1度に1つ使用するのですむのなら，1種類ずつ要求し次の要求をする前にその資源を解除すべきである．

- 可能なかぎり“共用”制御の要求をすること．図8-41のENQ マクロ命令で共用要求がなされていると，インタロックは生じない．ただし，資源の修正や変更を必要とする場合にも共用制御の要求をしろという意味ではない．資源に対する諸要件を注意深く分析し，共用要求で十分間に合う資源に対して排他的要求をすることがないようにという意味である．

- ENQ マクロ命令では1度に1つ以上の資源の制御要求を指定することができる．要求を出したプログラムは，要求資源のすべてが利用可能状態になるまで待ち状態におかれる．これらの資源は直ちに他のプログラムでは使用できなくなり，要求を出したプログラムで排他的に利用可能となる．

```
1) 2つの資源に別々の要求を出す例
      ENQ   (NAME 1 ADD, NAME 2 ADD, E, 8, SYSTEM)
      ENQ   (NAME 3 ADD, NAME 4 ADD, E, 10, SYSTEM)
2) 2つの資源に1つの要求を出す例
      ENQ   (NAME 1 ADD, NAME 2 ADD, E, 8, SYSTEM,
                  NAME 3 ADD, NAME 4 ADD, E, 10, SYSTEM)
```

図 8-42　資源の要求の仕方

図 8-42 を参照して欲しい．2つの ENQ マクロ命令を書く代りに，2) で示しているように1個の ENQ マクロ命令でコーディングすることができる．図 8-41 の場合の要求をこのような方法で行なうとインタロックを回避できる．一方のタスクで資源要求を行なうまえに，すべての資源要求が処理されるからである．なお，DEQ マクロ命令でも同様の書き方をして一度にすべての資源を解放すべきである．

- ある資源の利用が必ず別の資源の利用に依存しているような場合だと，一組の資源を1つの資源として END と DEQ マクロ命令で定義することができる．この方法は，いつでも組合せで使用される資源であるなら何種類になっても適用できる．しかし，これらの資源を独立して別々に要求すると，何らの資源保護も行なわれない．したがって，この種の資源は必ずセットにして要求すべきである．
- 沢山のタスクで同じ資源グループを利用し，しかもいくつかのタスクで最初の資源の制御権を保持している間につぎの資源の制御を要求するようなケースもまれではない．このような場合でもインタロックを回避することは可能だ．この場合には，資源の制御権を要求する順序をどのタスクでも同じにすることである．たとえば，資源A，B，Cが沢山のタスクを遂行するのに必要とすると，これらの資源の制御権の要求をいつもA，B，Cの順序で行なう．このようにすると，資源Aの要求が常時資源Bの要求に先行するので，インタロック状態は発生しない．

以上のような考慮を払うことによって，インタロックという最悪な状態を避けることができるが，これだけで完全とはいえない．場合によっては，ENQ

マクロ命令のオペランドで RET＝USE を指定することによって資源の制御権をくり返し要求する方法も考えられる．この方法だと，タスクを待ち状態におくことを防ぐことができるので，インタロック状況は回避できる．しかし，無駄な実行時間を伴なうのも事実だ．いずれにしろ，あらゆる場合のインタロック問題を解決するのには，資源を利用するすべてのユーザー間での協力関係を必要とするのである．

⑥ENQ/DEQ ルーチンの機能構造

すでに説明してきたように，ENQ/DEQ マクロ命令では逐次使用するシステム資源を qname と rname の対で表現する．その結果，qname と rname に対応して**主待行列制御ブロック**（Major Queue Control Block；略して**主 QCB**）と**副待行列制御ブロック**（Minor Queue Control Block；略して**副 QCB**）とが作られる．ただし，同一 qname と rname の資源の要求が出されている場合だと，すでに対応する主 QCB と副 QCB が作られているので，新たな QCB を作ることはしない．いずれにしろ，これらの資源に対して出された使用要求は**待行列要素**（Queue Element；略して **QEL**）に登録される．QEL は，どのタスクから資源の使用要求が出されたかを表示するものであり，対応するQCB からチェーニングされている．

主 QCB と副 QCB の形式と内容を示したのが**図 8-43** である．主 QCB には

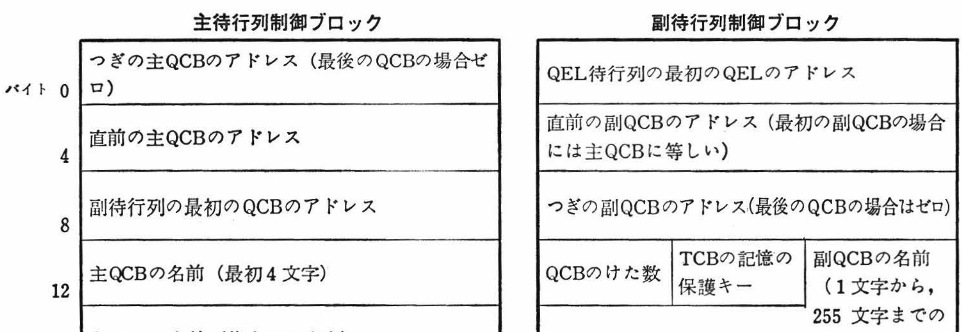


図 8-43　待行列制御ブロック（QCB）の形式と内容

バイト		
0	SMC	1 つぎの QEL のアドレス（最後の QEL の場合ゼロ）
4	CODE	5 直前の QEL のアドレス（QEL 待行列の最初の QEL の場合には副 QCB のアドレス）
8	QELTJID 1	9 ENQ マクロ命令が出されたプログラムの TCB アドレス
12	QELTJID 2	13 ENQ ルーチン用の SVRB のアドレス

図 8-44　待行列要素（QEL）の形式と内容

QCB をチェーニングするためのポインターと主 QCB の名前が収容されている．一方，副 QCB には副 QCB の相互間および QEL をチェーニングするためのポインターと副 QCB の名前が収容されている．

図 8-44 で示したのが QEL の形式と内容である．QEL には QEL 待行列をチェーニングするためのアドレスおよび資源要求を出したプログラムの TCB アドレスが収容されている．また，ENQ ルーチンを遂行するさいに必要となる SVRB のアドレスが入っている．さらに，4 種類の標識用バイト（SMC, CODE, QELTJID 1, QELTJID 2）がある．SMC は，該当 QEL が “must complete” 状況の要求を意味しているかどうかを表示するのに用いる．X ‘20’ だと “system must complete” 要求を，X ‘10’ だと “step must complete” 要求を表わす．また，X ‘00’ は “must complete” 状況が要求されていないことを示す．CODE バイトでは，共用要求か排他的要求かを表示する．ビット 0 が 1 にセットされていてと共用要求を，0 にセットされていると排他的要求を示す．QELTJID 1 と QELTJID 2 とはタイムシェアリング機能を用いたときの特別な表示に使用される．

主 QCB，副 QCB および QEL の相互関係を図式化すると**図 8-45** のようになる．図中の矢印はポインター（アドレス）を示している．主 QCB，副 QCB，それに QEL の各組み合せで特定タスクでの要求資源が表わされる．QELは，その資源に対して要求がなされた順にチェーンされる（図中の QEL 待行列参照）．副 QCB で直接ポイントされている QEL はいつでもその資源を使用しているプログラムを表示している．ところが，待行列の2番目以降の QEL は

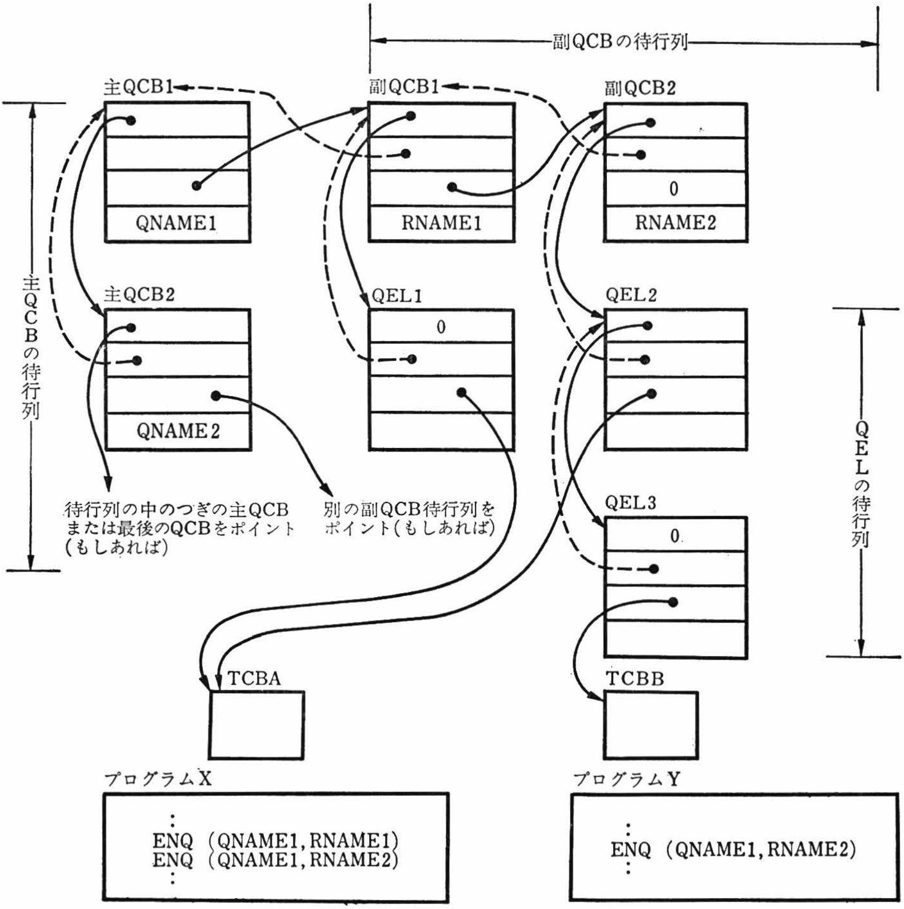


図 8-45　資源の待行列

その資源を使用中のこともあるし，待っていることもある．これについては後述する．図8-45ではプログラムXが2つの資源（QNAME 1，RNAME 1）と（QNAME 1，RNAME 2）とを使用していることを示している．一方，プログラムYが資源（QNAME 1，RNAME 2）を使用できるかどうかはプログラムXおよびプログラムYが出したENQマクロ命令のタイプによる．プログラムYが資源（QNAME 1，RNAME 2）を使用できず，プログラムXの使用が終るのを待っている間は要求ブロック（RB）の待ちカウント欄に1が加算さ

状　　　態	処理の方法
状態 1：資源待行列の最高位に共用要求のグループがある． 共用要求 → 共用要求 → 排他的要求	該当資源は，共用要求を出したプログラムでタスクの優先順位にもとづいて共用される．一方，排他的要求は，共用要求プログラムでの資源利用が完全に終わり，それらの要求が待行列から除去されるまで待たされる．
状態 2：排他的要求が待行列の最高位にあり，共用要求グループがそれに続いている． 排他的要求 → 共用要求 → 共用要求	排他的要求を出したプログラムでその資源を利用する．共用要求を出したプログラムは資源が解放されるまで待たされる．そのあと，タスクの優先順位にもとづいてその資源が共用される．
状態 3：排他的要求が待行列の最高位にあり，別の排他的要求がそれにつづいている． 排他的要求 → 排他的要求	最初の排他的要求プログラムがその資源を利用する．その間 2 番目の要求は待たされる．最初の要求プログラムの資源利用が完了すると，2 番目の排他的要求プログラムがその資源を利用する．
状態 4：共用要求グループが待行列の最高位にあり，排他的要求がそれに続いている．排他的要求に続いて別の共用要求グループがある． 共用要求 → 共用要求 → 排他的要求 → 共用要求 → 共用要求	まず，待行列の最高位にある要求を出した共用要求プログラムの間でタスクの優先順位にもとづいてその資源が共用される．排他的要求プログラムは，共用要求プログラムでの資源利用が完全に終わり，それらの要求が待行列から除去されるまで待たされる．そのあとで，その資源の排他的アクセスを行なう．待行列の下位にある共用要求はその資源が利用可能になるまで待たされる．それから，タスクの優先順位にもとづいてその資源を共用することとなる．

図 8-46　共用要求と排他的要求の処理の仕方

れ，待ち状態におかれている．

すでにみてきたように，逐次使用するシステム資源に対する要求のタイプには“共用”要求と“排他的”要求とがあり，ENQ マクロ命令のパラメータで指定する．同一システム資源に対して，2 つ以上の要求が出されている場合，つまり，1 つの副 QCB に対して 2 つ以上の QEL がチェーニングされている場合に，それらのうちのどの要求までが満足されるかは，“共用”と“排他的”との組み合せの状況いかんによる．“共用”と“排他的”要求とのさまざまな組み合せにおける資源の使用状況を示すと**図 8-46** のようになる．したがって，図 8-45 の場合，資源（QNAME 1，RNAME 2）に対してプログラム

XとYの双方で共用要求を出していると，プログラムYも資源（QNAME 1，RNAME 2）を使用できる状態にあり，プログラムXの使用完了を待つ必要はない．ところが，どちらか一方で排他的要求がなされていると，プログラムXでの資源利用が終るまでプログラムYは必然的に待ち状態に置かれざるを得ない．

ENQ ルーチンの処理

ENQ マクロ命令が出されたとき，ENQ ルーチンが遂行する処理の仕方は要求資源が使用されていない場合と使用中である場合とでは大きく異なる．前者の場合だと，QCB が存在しない場合があるので必要に応じて QCB を作成したうえで，所定の位置に QEL を挿入するのが普通である．後者の場合には，該当資源に対する QCB はすでに存在していることが前提になる．したがって，QCB を作成することはない．

ENQ マクロ命令では，RET パラメータを用いて条件つき要求をすることができた．したがって，無条件要求と条件つき要求とでも ENQ ルーチンでの処理内容が異なることになる．しかも，条件つき要求のタイプには TEST, USE, HAVE などがあるので，それぞれによって処理が異なる．

以上の要素を考慮に入れて ENQ ルーチンの処理の仕方をまとめると，**表**

表 8-13　要求資源が使用されていない場合の処理

RET パラメータ	RETパラメータの意味	QCB と QEL の作成および待行列への挿入	戻りコード	戻りコードの意味
TEST	要求プログラムが資源を即座に利用することができるかどうかを判定するために，待行列をテストする．制御ブロックは作成しない．	No	0	資源は利用可能である．
USE	要求プログラムが資源を即座にアクセスすることができるときのみ待行列に QCB と QEL を挿入する．	Yes	0	資源は利用可能である．
HAVE	資源利用の遅れは我慢できる．待行列に QCB と QEL を挿入する．	Yes	0	資源は利用可能である．
省略	戻りコードを作らないことを除いて HAVE と同じ．	Yes	コードなし	

表 8-14　要求資源が使用中である場合の処理

直前の QEL と現行要求のタイプ	RET パラメータ	処理内容
1. 待行列の直前のQELが "排他的" であるか，現行要求が "排他的" である	USE または TEST	戻りコード4をセットし，エグジット・ルーチンとディスパッチャーとを経由して，要求プログラムに制御権をもどす．その要求を示すための QEL は作成されない．
	HAVE または省略	SVRB 中の待ちカウントを増やして要求プログラムを待ち状態に置き，QEL を作成して QEL 待行列に入れる．また，タスクの切換えが必要であることを表示したうえで，タスクの切換えを行なうためにディスパッチャーにブランチする．なお，HAVE パラメータが指定されていると戻りコードは4にセットされる．
2. 待行列の直前の QEL と現行要求の双方が "共用" であるか，またはその資源に対する QEL が存在しない．	TEST	戻りコード0をセットし，エグジット・ルーチンとディスパッチャーとを経由して，要求プログラムに制御権をもどす．QEL は作成されない．
	省略	新しい QEL を作って QEL 待行列に入れる．そのあとエグジット・ルーチンとディスパッチャーを経由して要求プログラムに制御権をもどす．
	USE または HAVE	戻りコード0をセットし，新しい QEL を作って QEL 待行列に入れる．そのあとエグジット・ルーチンとディスパッチャーを経由して要求プログラムに制御権をもどす．

8-13 と**表 8-14** のようになる．ENQ マクロ命令を実行した時点で，要求資源が使用されていない場合の処理をまとめたのが表 8-13 である．一方，要求資源が使用中である場合の処理は表 8-14 で示してある．

ENQ ルーチンは，表 8-13 と表 8-14 で示している処理以外に以下で示す5種類の処理を遂行する：

① 次の条件が満足されると，"must complete" 状況の TCB 標識をセットする．

- ENQ マクロ命令を出したプログラムの記憶保護キーが0である．
- ENQ マクロ命令の RET オペランドが TEST 以外である．
- SMC オペランドが指定されている．
- ENQ ルーチンが用いる SVRB の待ちカウントが0である．

以上4つの条件が満足されたとき，**表 8-15** で示した処理を行なうのである．

表 8-15　タスクが "Must Complete" 状況にあるときにセットされるTCB 標識

共通名	記号名（TCB の中の位置）	標識がセットされるTCB	セットされた標識の目的
"Must Complete" ディスパッチ不能標識（システムまたはジョブ・ステップ）	TCBSYS（33バイト目の第4ビット）	"must complete"TCBを除いたシステム内の全TCB，およびある種のシステム・タスクの TCB	該当 TCB に関連するすべてのルーチンを実行させてはならない旨をディスパッチャーに知らせる．
	TCBSTP（33バイト目の第5ビット）	"must complete"TCBを除いたジョブ・ステップ内の全 TCB	
"Must Complete" 標識（システムまたはジョブ・ステップ）	TCBFSMC（30バイト目の第3ビット）	"must complete"TCB	該当タスクがエラーであるとき，そのタスクを終了させるべきである旨を ABEND ルーチンに知らせる．
	TCBFJMC（30バイト目の第4ビット）		
非同期エグジット禁止標識	TCBFJMC（29バイト目の第7ビット）	"must complete"TCB	該当タスクに対するユーザー・エグジット・ルーチンをスケジュールしてはならない旨をステージ3エグジット・エフェクターに知らせる．

"must complete" の機能は，ジョブ・ステップ内あるいはシステム内の他のタスクを**ディスパッチ不能**（non-dispatchable）にすることである．これにより，処理が早くなるとともに別のタスクが原因で異常終了することがなくなる．

② インプット・パラメータで指定されたアドレスを**妥当性検査ルーチン**（validity check routine）でチェックし，誤りのある時は該当タスクを異常終了する．ENQ マクロ命令を出したプログラムが記憶保護キー0で働いている場合だとこのチェックは行なわない．

③ すでに ENQ をかけている資源に対して再び ENQ マクロ命令を出した場合は，RET パラメータの値によって次の処理を行なう．

- RET パラメータが省略されている場合：異常終了させる．
- RET パラメータが指定されている場合：戻りコード8を返す．

④ **ロールアウト不能**（nonrolloutable）カウントを1増やす．

⑤ **共用 DASD**（shared DASD）に対して RESERVE マクロ命令が出され

た場合は，通常の ENQ の外に UCB の中の**予約カウント**（reserve count）を1増やす．

上記のような処理を行なったあとで ENQ ルーチンからリターンする場合のやり方は，要求システム資源が直ちに使用できるか，または使用する順番が来るのを待つかによって異る．

要求資源が直ちに使用できる場合には，エグジット・ルーチンにブランチして，SVRB を RB 待行列から除去してそのスペースを解放する．ディスパッチャーは要求プログラムの RB の中の旧 PSW を CPU にロードして，ENQ マクロ命令を出したプログラムを再開させる．

一方，要求資源が使用できるまで持つ場合だと，SVRB の中にある**次の命令アドレス**（next instruction address）が CVT にある SVC 3 の命令をポイントするように変更する．そのあとで，タスクの切換が必要なことを所定の場所に記録したうえでディスパッチャーにブランチする．なお，SVC 3 はエグジット・ルーチンを遂行するための割込み命令である．

DEQ ルーチンの処理

ENQ をかけたシステム資源の使用が終了した旨を知らせるのには DEQ マクロ命令を使用する．これにより監視プログラムの中の DEQ ルーチンが機能することになる．DEQ ルーチンは次に示す7個の処理機能を果す：

① DEQ するシステム資源から QEL を除去し，その資源に対する QEL のチェーニングを更新する．その結果，ポイントすべき QEL が存在しない副 QCB が生じると，その副 QCB を主 QCB からのチェーニングからはずし，そのスペースを解放する，さらに，副 QCB を除去した結果，ポイントすべき副 QCB がなくなった主 QCB ができた場合には，それも主 QCB の待行列からはずしスペースを解放する．

② QEL 待行列のチェーニングを更新したために新しく活動可能状態となななタスクが生じることが予想される．そのために**図 8-47** で示したような処理を遂行する．

状　　態	処 理 の 方 法
状態1："共有"QELが除去され，QEL待行列の最高位に新しい"共用"QELがある． ●除去されたQEL　"共有QEL" ●新しい最高位のQEL　"共有QEL"	要求プログラムに対応するRBの中の待ちカウントから1を引かない（すでに要求プログラムには該当資源のアクセスが許されているから）．
状態2："共有"QELか"排他的"QELのどちらかが除去され，QEL待行列の最高位には"排他的"QELがある． ●除去されたQEL　"共有"または"排他的"QEL ●新しい最高位のQEL　"排他的"QEL	要求プログラムに対応するRBの中の待ちカウントから1を引く．その結果，待ちカウントがゼロになったら，呼び出しプログラム（つまり，DEQマクロ命令を出したプログラム）の代りに要求プログラムをディスパッチできるかどうかテストするためにタスク・スイッチング・ルーチンを呼ぶ．
状態3："排他的"QELが除去され，QEL待行列の最高位には"共有"QELがある． ●除去されたQEL　"排他的"QEL ●新しい最高位のQEL　"共用"QEL	要求プログラムに対応するRBの中の待ちカウントから1を引く．その結果，待ちカウントがゼロになったら，タスク・スイッチング・ルーチンを呼び出す．新しい最高位のQELは，"共有"グループの最初のQELであるので，そのグループの他のQELに対して同じ手順をくり返す．

図 8-47　つぎの待ち状態の資源要求プログラムに対する処理の方法

図8-47で示した処理の結果，あるタスクのSVRBの待ちカウントが0になると**タスク切換ルーチン**（task switching routine）にブランチし，新たに活動可能状態になったタスクにCPUの制御権を渡すかどうかを決定する．このタスクが次にCPUを使用することが決まると，中核プログラムの中にある新TCBポインターを0にする（新TCBポインターの機能に関しては「監視プログラムの内的サービス機能」の項で説明）．

③ DEQマクロ命令で"reset must complete"パラメータを指定している場合は，"set must complete"の機能でTCBにセットされた標識をもとに戻す．

④ インプット・パラメータで指定されたアドレスを妥当性検査ルーチンでチェックし，誤りのある時は該当タスクを異常終止させる．ただし，DEQマクロ命令を出したプログラムが記憶保護キー0で働いている場合にはこ

表 8-16　資源利用が完了したときのエラー状態

状　　態	RET パラメータ	処　理　内　容
資源の名前が QCB 待行列の中に見いだせない，または呼び出しプログラムの TCB アドレスを収容した QEL が見い出せない.	HAVE	①戻りコード 8 をセットし，該当資源が待行列に入れられていないことを表す．残りのパラメータ・リストを処理したあと，エグジット・ルーチンとディスパッチャーを経由して制御権を呼び出しプログラムに返す.
	省　　略	②エラー・コード 130 をセットし，呼び出しプログラムのタスクを異常終止させるために ABENDルーチンの連係を行なう.
呼び出しプログラムの TCB アドレスを収容した QELは見つかったが，その QEL が QEL 待行列の最高位に位置していない．または，待行列の最高位の“共有” QELグループの 1 つでない	HAVE	戻りコードが 4 であることを除いて①の場合と同じ．呼び出しプログラムのタスクでは該当資源のアクセスがなされないことを示す.
	省　　略	エラー・コードが 230 である点を除いて②の場合と同じ.

のチェックは行なわれない.

⑤ DEQ しようとしている資源に関するチェーニング，つまり主 QCB から副 QCB と副 QCB から QEL へのポインターがあり，そのタスクの QEL が QEL 待行列の最高位にあるかどうか調べる．これに該当しないと**表 8-16** で示した処理を行なう.

⑥ ロールアウト不能カウントを 1 だけ減ずる.

⑦ DEQ をかけた QEL に予約フラグが立っている場合には，共用 DASD に対して出された RESERVE マクロ命令に対する DEQ であることがわかる．予約フラグは，QEL の CODE 欄のビット 1 で示されている．この場合には，UCB の予約カウントを 1 だけ減算し，このカウントが 0 になると該当 DASD に対して EXCP 命令を使って**解除指令**(release command) を出す.

以上が DEQ ルーチンの処理内容である.

4）ユーザー・エグジット・ルーチンのスケジューリング機能

予知できない事象を処理するために，ユーザー・プログラムでエグジット・ルーチンの実行をあらかじめ要請しておくことができる．予知できない事象としては，タスクの終了状態，一定の時間間隔の経過，特殊な入出力処理（テープ・ラベルの検査や入出力エラーの検査）などがある．**ユーザー・エグジット・ルーチン**（user exit routine）はしばしば**非同期的エグジット・ルーチン**（asynchronous exit routine）と呼ばれる．ユーザー・プログラムの実行と同期して実行されるルーチンでないことから非同期的といわれるのである．ともあれ，ユーザー・エグジット・ルーチンのスケジューリングは4種類の監視プログラム・ルーチンで処理される．ステージ1エグジット・エフェクター（stnge 1 exit effector），ステージ2エグジット・エフェクター（stage 2 exit effector），ステージ3エグジット・エフエクター（stage 3 exit effector）およびエグジット・ルーチン（exit routine）の4種類がこれに当る．ただし，これらのルーチンではユーザー・プログラム・チェック・ルーチンの実行のスケジューリングは行なわない．プログラム割込みの異常終止処理は，SPIE マクロ命令でインターセプトされるからである．

ユーザー・エグジット・ルーチンのスケジューリングの流れを示したのが**図8-48**である．これにもとづいて監視プログラム・ルーチンがどのように機能するか説明しよう．

ユーザー・エグジット・ルーチンの実行要請を処理するためには，いくつかの手順が必要であり，しかも図8-48で示しているように他のタスクのプログラムの実行とからみあって進んで行く．その手順は，ユーザー・プログラムでエグジット・ルーチンの要求を出すことによって開始される．その要求はATTACH, STIMER, DCB などのマクロ命令のオペランドの指定を通じてなされる．

ATTACH ルーチンのようなシステム・ルーチンは，必要に応じて**割込み待行列要素**（Interruption Queue Element；略して **IQE**）を作成する．IQE の形式と内容は**図8-49**に示している通りであり，ステージ2と3のエグジット・エフエクターとエグジット・ルーチンがユーザー・エグジット・ルーチンをス

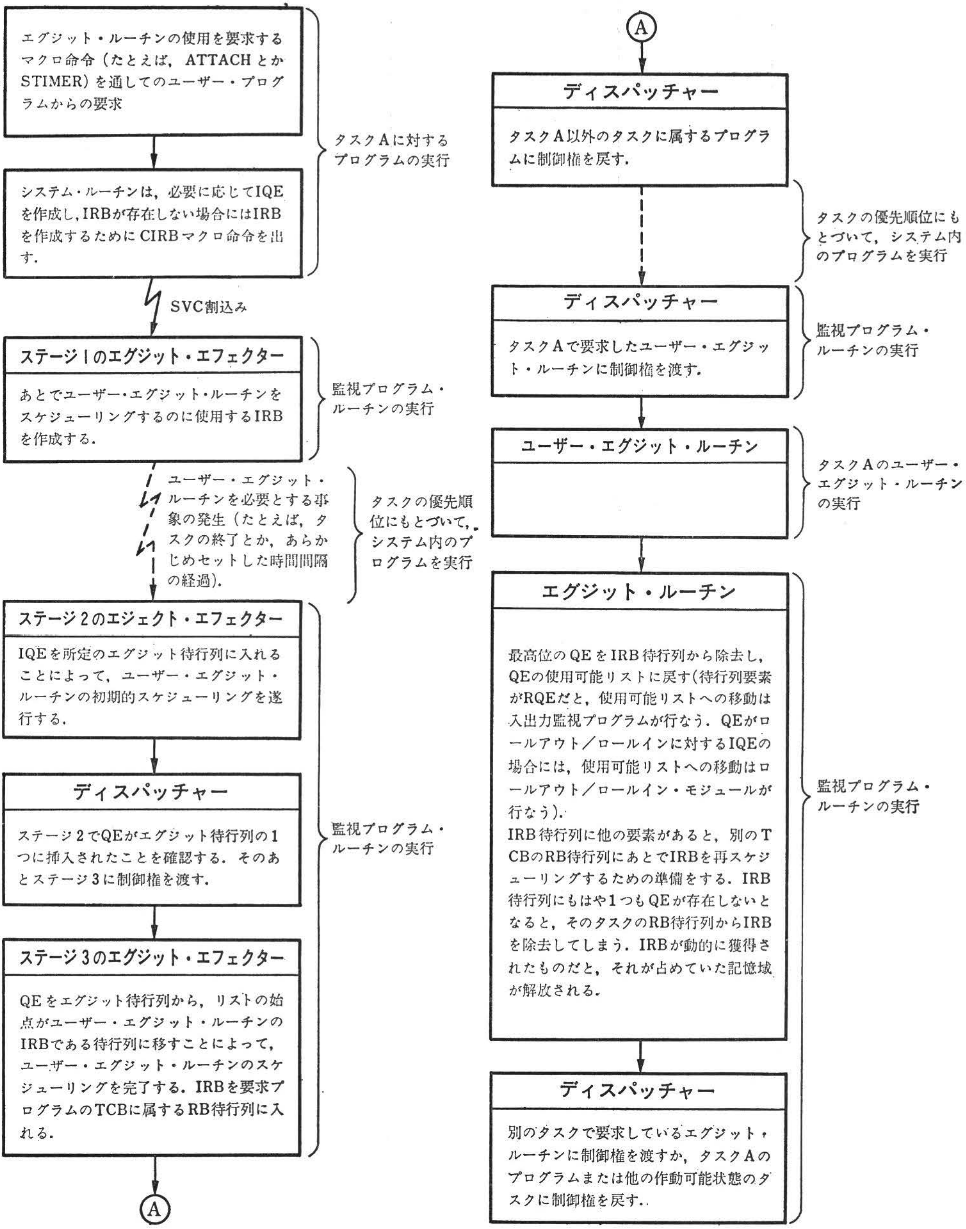


図 8-48　ユーザー・エグジット・ルーチンのスケジューリング

	バイト				
	0	予約フィールド	1 IQELNK：IQE待行列上のつぎのIQEのアドレス	4 IQEPARAM：非同期的エグジット・ルーチンに受け渡すパラメータ	
ロールアウト/ロールインのパラメータ・リスト(オプショナル)	8	予約フィールド	9 IQEIRB：該当要求のためにスケジュールされるIRBのアドレス	12 予約フィールド	13 IQETCB：該当要求に関連するTCBのアドレス
	16	RPLTCB：領域の拡張を要求したり，解除したりするタスクのTCBアドレス		20 予約フィールド	21 RPLSZPQE：ロールアウト要求の領域のサイズ，またはロールイン要求のさいのエリアを記述しているPQEのアドレス

図 8-49　割込み待行列要素（IQE）の形式と内容

ケジュールするために使用する．ただし，データ管理ルーチンは IQE を作成しない．というのは，入出力待行列要素がすでに存在しているからである．これら要素は一般に**要求待行列要素**（Request Queue Element；略して **RQE**）と呼ばれ，入出力監視プログラムが作成する．ともあれ，システム・ルーチンはそのあとで特殊なシステム・マクロ命令である CIRB を出す．CIRB マクロ命令は，ステージ1エグジット・エフエクターに**割込み要求ブロック**（Interruption Request Block；略して **IRB**）を作らせる．IRB の形式と内容は**図 8-50**に示している通りであり，あとでユーザー・エグジット・ルーチンをスケジュールするのに使用される．

ステージ1エグジット・エフェクターがユーザー・ルーチンを表わす IQE を作成しても，そのルーチンを必要とする事象が発生するまで何らのスケジューリングも行なわれない．しかし，予知できない所定の事象が発生すると，ステージ2エグジット・エフェクターが機能する．監視プログラム・ルーチンの1つであるステージ2エグジット・エフェクターは，さきに作成した IQE ないしは RQE を所定のエグジット待行列に入れることによって，ユーザー・エグジット・ルーチンの初期的スケジューリングを達成する．特定のエグジット・ルーチンの使用要求を表示するための待行列には2つの種類がある．1つは IQE を収容した待行列であり，タイマー・エグジット・ルーチンやタスクの終了ルーチンのたぐいのルーチンの使用要求を表示する．もう1つは RQE だけを収容する待行列であり，データ管理用エグジットの要求を示す．この RQ

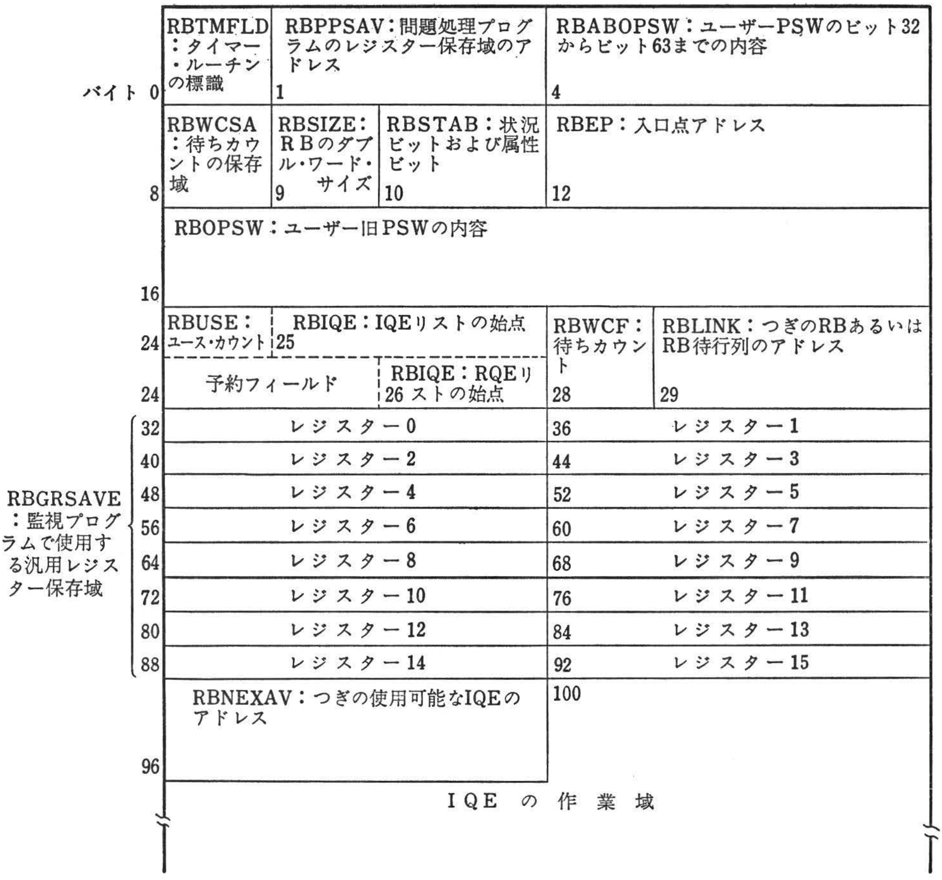


図 8-50　割込み要求ブロック（IRB）の形式と内容

Eは入出力監視プログラムが入出力要求をスケジュールする要素と同じである．2種類のエグジット・ルーチンはいずれも，同一エグジット・ルーチンを要求するプログラムのタスク優先順位とは無関係に，先入れ－先出しの順序で用いられる．**待行列要素**（Queue Element；略して**QE**），つまりIQEかRQEかいずれかのエグジット待行列に置かれても，エグジット・ルーチンそのものを表示するIRBがいまだタスクのRB待行列に入れられていないので実行す

ることはできない．したがって，ステージ２エグジット・エフェクターが QE を所定のエグジット待行列におく作業はいわば予約的仕事にすぎないといえる．

ユーザー・エグジット・ルーチンのスケジューリングを完成させるのは，ステージ３エグジット・エフェクターである．ディパッチャーのサブルーチンともいえるステージ３エフェクターは，先述のエグジット待行列から所定の QE を取り出して，エグジット・ルーチンを表示している別の待行列に挿入する．これで特定のエグジット・ルーチン用の QE となる．さらに，ステージ３エフェクターは IRB を要求プログラムのタスクに属する RB 待行列に入れる．これによって，ユーザー・エグジット・ルーチンのスケジューリングが完結する．これでユーザー・エグジット・ルーチンは他のプログラムと同様に CPU 時間を競い合うことができる状態となる．したがって，IRB をポイントしている要求プログラムのタスクが作動可能状態の中で最も高い優先順位を持っているとディスパッチャーは IRB の中の旧 PSW をロードしてユーザー・エグジット・ルーチンを実行させるのである．

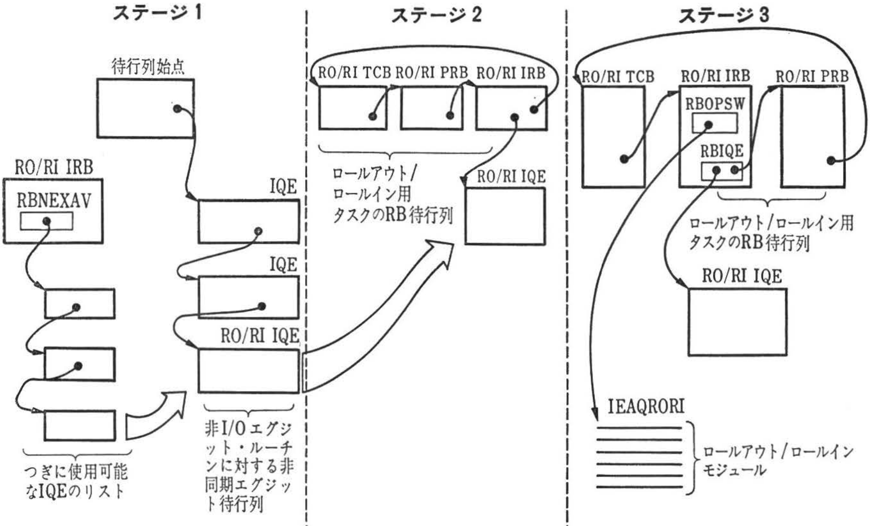
ユーザー・エグジット・ルーチンの実行が終ると，監視プログラムのエグジット・ルーチンに制御権が渡される．このエグジット・ルーチンは，QE で構成されている IRB 待行列の中の最高位に位置している QE を除去する．

以上の機能をまとめる目的で，ユーザー・エグジット・ルーチンをスケジュールするための制御ブロックのチェーニングの様子を示したのが**図 8-51** である．この例では，ロールアウト／ロールイン・タスクのスケジューリングを例にとっている．

以下でステージ１エグジット・エフェクター，ステージ２エグジット・エフェクターおよび監視プログラム・エグジット・ルーチンの機能をまとめておこう：

① ステージ１エグジット・エフェクター（CIRB ルーチン）

ステージ１エグジット・エフェクターは，ATTACH ルーチンのようなシステム・ルーチンで使用され，IRB, IQE のスペースを確保すると同時に IRB の



SHEDROルーチン(システム・ルーチン)はロールアウト／ロールインIQEを使用可能リストから得るか，またはスペースを確保し新しいIQEを初期値化することによって得る．そのあとステージ2のエグジット・エフェクターを呼び出し，そのIQEを待行列に入れる．

ロールアウト／ロールイン用IQEはエグジット待行列から取除かれ，ロールアウト／ロールインIRBからポイントされるように待行列に入れられる（これは，ステージ3のエグジット・エフェクターで遂行される）．

ステージ3のエグジット・エフェクターは，ロールアウト／ロールインのRB待行列を並べかえることによって，そのタスクのTCBを作動可能にする．その結果，ロールアウト／ロールイン用TCBは作動可能なロールアウト／ロールイン用IRBをポイントする．ロールアウト／ロールイン用タスクはいまや作動可能状態にあるので，ディスパッチャーはこのタスクの制御権をロールアウト／ロールイン・モジュールの中のIEAQRORIルーチンに受渡す．

《凡例》
RO/RI　：ロールアウト／ロールイン
RBOPSW：RBの旧PSW
───→：ポインター

図 8-51　ロールアウト／ロールイン用タスクのスケジューリングのステップ

初期値化を遂行する．

ステージ1エグジット・エフェクターがシステム・ルーチンで使用される場合，IRB と IQE のスペースは**システム待合せ域**（system queue area）からとる．また，レジスター保存域はサブプール0から獲得する．そのあとで，IRB の中の RBGRSAVE, RBSIZE, RBEP, RBABOPSW の各フィールドを初期値化する．

IRBの初期値が終ると，監視プログラム・エグジット・ルーチンとディスパッチャーを経由して，優先順位の最も高いタスクの処理プログラムに制御権を移す．

②　ステージ2エグジット・エフェクター

ユーザー・エグジット・ルーチンの実行をスケジューリングするための前提条件である事象が発生すると，ステージ2エグジット・エフェクターに制御権が渡る．ステージ2エフェクターは，タスク終了ルーチンや**タイマー第二次割込み処理ルーチン**（timer second-level intrruption handler）などのサブルーチンとして使用される．

ステージ2エグジット・エフェクターの機能は，レジスター1を使ってインプット・パラメータとして渡されたIQE またはRQE のアドレスをもとに，QE をエグジット待行列に入れることである．レジスターの値が正である場合には，パラメータがRQE であると判断して，RQE エグジット待行列に挿入する．先述したように，RQE エグジット待行列はデータ管理用のエグジットをスケジュールするためのものである．一方，レジスター1の値が負の場合には，パラメータがIQE を指すものであると判定して，IQE エグジット待行列に入れる．この待行列はデータ管理以外のエグジット・ルーチンをスケジュールするための待行列である．

このあと，ステージ2エグジット・エフェクターはステージ3スイッチをセットする．このスイッチはディスパッチャーによってチェックされ，オンであるとステージ3エグジット・エフェクターを呼び込みユーザー・エグジット・ルーチンをスケジュールする．

③　ステージ3エグジット・エフェクター

ディスパッチャーがステージ3スイッチがセットされているのを知ると，ステージ3エフェクターに制御権を渡し，IQE 待行列ないしはRQE 待行列にあるQE を処理する．ステージ3スイッチがセットされていることは，ユーザー・エグジット・ルーチンをスケジュールする必要のある事象が発生し，それがまだ処理されていないことを意味する．そして，エグジット待行列にあるIQE

ないしは RQE はその事象が発生したことを表わしている.

ステージ3エグジット・エフェクターの行なう処理は，つぎの3段階に分けられる：

- IQE を IQE エグジット待行列から除去し，IRB 待行列に入れる.
- IRB を TCB からの RB チェーンの先頭におく.
- IRB をチェーンに加えた結果，そのタスクが作動可能状態になった場合にはタスク切換ルーチンに行く．新たに作動可能状態になったタスクの優先順位が，現在 CPU を使用しているタスクのそれよりも高い場合は中核プログラムの中にある新 TCB ポインターに新たに作動可能となったタスクのアドレスを入れる.

上記の3段階を IQE エグジット待行列上のすべての IQE について行なう．ただし，IRB がすでに TCB からの RB チェーンに存在する場合だとステップの2と3は行なわない.

IQE エグジット待行列の処理が終ると，RQE エグジット待行列上の RQE を処理する．この処理は，システム・エラー処理ルーチンをスケジュールする場合を除いて IQE の処理と同様である.

④ 監視プログラム・エグジット・ルーチン

ユーザー・エグジット・ルーチンで RETURN マクロ命令を出すと，SVC FLIH を経由して監視プログラム・エグジット・ルーチンに制御権が渡る．この場合，IRB がポイントしている要求が IQE か RQE かにより処理方法が異なる.

IRB が IQE をポイントしていると，つぎの処理が行なわれる：

- IRB がポイントしている IQE のうち先頭の IQE を待行列からはずす.
- 待行列からはずしたIQE の処理は，実行されたユーザー・エグジット・ルーチンのタイプによって異なる．たとえば，ロールアウト／ロールインのルーチンを実行した場合だと，その IQE を**つぎに使用可能なリスト**（next available list）に加える.

- 先頭の IQE を除去してもまだ IQE が IRB 待行列にある場合には，IRB をチェーンからはずさないで再びユーザー・エグジット・ルーチンをスケジュールさせる．たとえば，2つ以上のサブタスクを ATTACH して，それらのサブタスクが同一エグジット・ルーチンを使用している場合に，それらのサブタスクがほとんど同時に RETURN すると同一エグジット・ルーチンに対する IRB に対して複数個の IQE がチェーンされることになる．IRB がポイントしている IQE がなくなった場合だと，IRB を TCB の RB 待行列から除去する．
- IRB の中のユース・カウントからマイナス1する．ユース・カウントが0になったら IRB のスペースを解放する．

一方，IRB が RQE をポイントしている場合には，IQE の場合とちがい使用が終った RQE は入出力監視プログラムが使用する RQE 待行列にもどす．IRB が動的に確保されたものであると（永久的 IRB でない場合）IRB のスペースは解放される．

5）タスクの非同期エグジット・ルーチンの指定機能

あらかじめ STAE マクロ命令が出されていると，タスクの異常終止が生じその処理に入ったとき，異常終止の処理をインタセプトし，ユーザーのエグジット・ルーチン（**STAE エグジット・ルーチン**という）で処理することができる．このために使用される STAE マクロ命令の一般形式は**表 8-17** に示してある．それぞれのパラメータの指定内容と意味は**表 8-18** にまとめてあるので

表 8-17　STAE マクロ命令の一般形式

［記　号］	STAE	{0 | エグジット・アドレス} [, OV | , CT] [, PARAM=リスト・アドレス] [, XCTL={YES | <u>NO</u>}] [, PURGE={<u>QUIESCE</u> | HALT | NONE}] [, ASYNCH={YES | <u>NO</u>}]

表 8-18　STAE マクロ命令のパラメーター一覧表

パラメータ	指定の内容	意味
エグジット．アドレス	アドレス	このマクロ命令を出したタスクが異常終止したときに実行すべき STAE エグジット・ルーチンのアドレス．ここに 0 を指定するともっとも新しく要求された STAE 要求が取消される．
OV		この STAE マクロ命令で受け渡されたパラメータで，以前の STAE 要求に含まれていたデータをオーバレイすることを表示．
CT		新規の STAE 要求を作りあげることを表示．OV，CT のいずれも指定されていないと CT がとられる．
PARM=	リスト・アドレス	STAE エグジット・ルーチンが用いるデータが収容されているパラメータ・リストのアドレス．
XCTL=	YES	XCTL マクロ命令が出されても，STAE マクロ命令が取消されない．
	NO	XCTL マクロ命令が出されると，STAE マクロ命令は取消される．このパラメータが省略されると，XCTL=NO とみなされる．
PURGE=	QUIESCE	STAE エグジットが取られたとき，入出力オペレーション要求の未処理のもののすべてを保存することを指示．STAE エグジット・ルーチンの終りで，未処理の入出力要求を処理する再試行ルーチンを書くことができる．
	HALT	STAE エグジットが取られたとき，入出力オペレーション要求の未処理のものがあっても保存しないことを指示．
	NONE	STAE エグジットが取られても，入出力処理はそのまま続行するように指示．
ASYNCH=	YES	STAE エグジット・ルーチンの処理を中断して，非同期的割込み処理ができるように指示．
	NO	同非期的割込み処理を許さないことを指示．

参照して欲しい．

STAE マクロ命令で指示した STAE エグジット・ルーチンではタスク終了の前処理を行ない，必要に応じて再試行ルーチンを実行させることによって，タスクの異常終止を回避することができる．

STAE マクロ命令が実行されると，監視プログラムの中の STAE サービス・ルーチンに制御権が渡る．STAE サービス・ルーチンでは，STAE マクロ命

バイト	内容	バイト	内容
0	標識：該当タスクに対する先行SCBに対応するTCB STAE標識のバイト．該当タスクの最初のSCBだとゼロにセットされる	1	先行SCBのアドレス：該当タスクに対する先行SCBのアドレス．このSCBが該当タスクに対して最初に作成されたものだとゼロが入る
4	STAEエグジット・ルーチンのアドレス：STAEマクロ命令で指示したユーザー作成のSTAEエグジット・ルーチンのアドレス		
8	標識：下記〈注1〉を参照のこと	9	STAEエグジット・ルーチンのパラメータ・リストのアドレス：STAEマクロ命令で指定したSTAEエグジット・ルーチンに受渡すべきパラメータ・リストのアドレス
12	STAE標識：下記〈注2〉を参照のこと	13	RBのアドレス：STAEマクロ命令を出したタスクの要求ブロックのアドレス

〈注1〉

ビット位置	表示内容
0〜4	予約ビット
5	非同期割込みを許可
6〜7	00—Quiesce I/O
	01—Halt I/O
	10—I/O介入のバイパス

〈注2〉

ビット位置	表示内容
0	XCTLが出されてもエグジット・ルーチンでSCBをキャンセルしない
1	ISAM/TAMスイッチ
2	STAI SCB
3	以前に使用ずみのSCB
4〜7	予約ビット

図 8-52　STAE 制御ブロック（SCB）の形式と内容

令の妥当性を検査し **STAE 制御ブロック**（STAE Control Block；略して **SCB**）を作成する．また，作成された SCB を該当タスクの TCB からポイントするためのアドレスを所定のフィールド（TCBNSTAE）に挿入する．SCB の形式と内容とは**図 8-52** に示す通りであり，STAE マクロ命令のオペランドで指定した内容の集約と SCB をチェーニングするためのアドレスとが収容される．

STAE マクロ命令が実行されたあと，該当タスクが異常終止すると TCB の中の TCBNSTAE フィールドでポイントしている SCB をみて，それがポイントしている STAE エグジット・ルーチンを実行する．STAE エグジット・ル

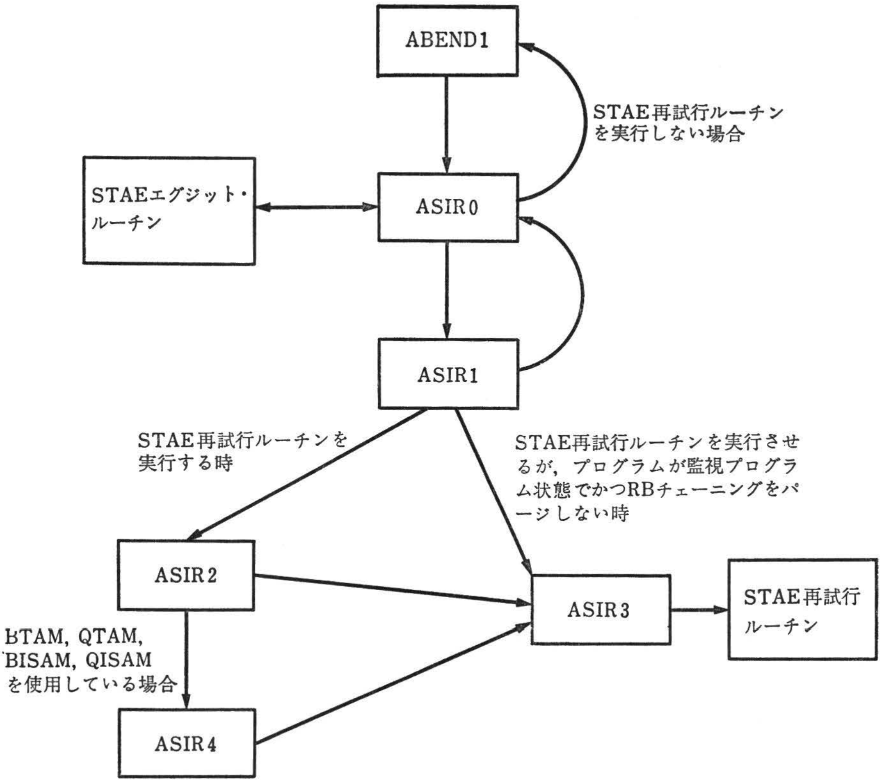


図 8-53　ASIR ルーチンの連係図

ーチンの戻りコードによっては再試行ルーチンを実行させ，タスクの異常終止を避けることもできる．STAE エグジット・ルーチンの実行にさいして機能する監視プログラムのルーチンが **ABEND/STAE インターフェース・ルーチン** (ABEND/STAE Interface Routine；略して **ASIR**) であり，5種類のルーチンから構成されている．これらルーチンの相互の関係を図示したのが図 8-53 である．この図をもとに各ルーチンの機能を要約するとつぎのようになる：

- ASIR 0： ABEND 1 ルーチンから制御権が渡される．STAE マクロ命令のPURGE オペランドで指示された入出力オペレーション要求の処理を行ない，必要な作業域を確保したうえで，ユーザー作成の STAE エグジッ

ト・ルーチンをスケジュールする.

- ASIR 1：　ASIR 0 から制御権が渡される. STAE 再試行ルーチンの実行要請がない場合には, ASIR 0 を経由して ABEND 1 ルーチンへ制御権を戻す. 再試行ルーチンの実行要請があると ASIR 2 を呼び出す. STAE マクロ命令を出したプログラムが監視プログラム状態にあり, しかも RB チェーニングをパージしないで再試行を行なうよう要求している場合には, ASIR 3 を呼び出す.
- ASIR 2：　ASIR 1 から制御権を受けとり, 異常終止したプログラムのデータ・セットを CLOSE する. BTAM, QTAM, BISAM, QISAM の DCB があると, ASIR 4 へ制御権を渡す. そうでないと ASIR 3 へ行く.
- ASIR 3：　ASIR 1, ASIR 2 または ASIR 4 から制御権を受けとり, STAE 再試行ルーチンをスケジュールする.
- ASIR 4：　ASIR 2 から制御権を受けとる. BTAM, QTAM, BISAM あるいは QISAM を用いているオープン状態のデータ・セットを探索し, これらのデータ・セットを CLOSE する. そのあとで ASIR 3 へ行く.

以上のことを念頭に図 8-53 をながめると, ASIR ルーチンの概略が把握できよう. 以下で各ルーチンの機能をもう少し具体的にしておこう：

① STAE サービス・ルーチン

先述したように, STAE サービス・ルーチンは STAE マクロ命令が実行されることによって制御権を得て, STAE マクロ命令のオペランドの妥当性を検査したあとで, SCB を作成する（後述するように修正することもある)。SCB を作成した場合のチェーニングは**図 8-54** に示したようになる. 注意すべきことは, STAE エグジット・ルーチンは **“後入れ-先出し法”** にしたがって機能す

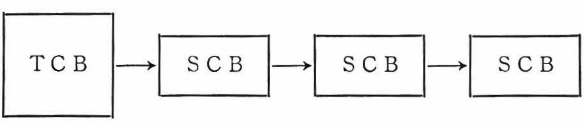


図 8-54　SCB のチエーニング

ることである．したがって，STAE マクロ命令によって新規の SCB が作られたときは SCB 待行列の先頭におかれる． それゆえに，最新の SCB がいつでも TCB の TCBNSTAE フィールドから直接ポイントされるようになる．

STAE マクロ命令のオペランドでは2種類のオプション（OV と CT）が指定できるので，オプションによって SCB の処理方法が異なる．CT（Create）だと，GETMAIN により16バイトのスペースをとり，そこに SCB を作りあげる．そのうえで，作りあげた SCB を SCB 待行列の先頭に位置づける．一方，OV（Overlay）の場合には，SCB 待行列の先頭にある SCB の STAE エグジット・ルーチンのアドレス・フィールド，パラメータ・リストのアドレス・フィールドおよび STAE 標識フィールドを STAE マクロ命令で指定した値でおきかえる．

② ASIR 0 ルーチン

ABEND 1 ルーチンが SCB があることを知ることによって， ASIR 0 に制御権を渡す． SCB 待行列の先頭の SCB から RB アドレス・フィールドをチェックし，RB 待行列の RB をポイントしている最初の SCB を用いて STAE エグジット・ルーチンを実行させる．

STAE マクロ命令で PURGE パラメータが指定されていると， パラメータに従った処理を行なうための入出力監視プログラムのルーチンを呼ぶ．そのあとで，**図 8-55** に示したような作業域を GETMAIN 命令で確保し， STAE エグジット・ルーチンに受渡す．この場合，作業域のアドレスをレジスター1に入れてエグジット・ルーチンに渡すのである．なお， ASIR 0 ルーチンは SYNCH マクロ命令を用いて RB を作り，その RB のもとで STAE エグジット・ルーチンを実行させる．

③ ASIR 1 ルーチン

STAE エグジット・ルーチンが 戻りコード 4 を返すと，STAE 再試行ルーチンを実行することを意味する． それによって， ASIR 1 ルーチンはつぎの様にパラメータ・レジスターを準備する：

バイト		
0	STAEエグジット・ルーチンのパラメータ・リスト・アドレスまたは0	異常終止（ABEND）完了コード
4		
8	異常終止時の PSW	
16	異常終止直前の問題処理プログラムの PSW	
24	異常終止時の汎用レジスター0～15の内容（64バイト）	

●問題プログラムで STAE マクロ命令を出した場合；

バイト		
88	異常終止プログラムの名前または0	
96	異常終止プログラムの入口点のアドレス	0

●監視プログラムで STAE マクロ命令を出した場合；

バイト		
88	異常終止プログラムの要求ブロックのアドレス	0
96	0	

図 8-55　STAE エグジット・ルーチンの作業域

- レジスター7：IOB 回復待行列の先頭の IOB アドレス
- レジスター8：作業域のアドレス
- レジスター10：STAE 再試行ルーチンのアドレス
- レジスター11：12：異常終止したプログラムの名前
- レジスター13：異常終止したプログラムの入口点アドレス

このあとで，先述した条件にしたがって，ASIR 2 か ASIR 3 のいずれかに制御権を渡す．

④　ASIR 2 ルーチン

ASIR 1 から渡されたパラメータ・レジスター7から13の内容を SVRB の特

定域に保存する．

異常終止したプログラムと STAE マクロ命令を出したプログラムとが異る場合には，STAE マクロ命令を出したプログラムの RB より RB 待行列上で先行している RB に対応したプログラムが OPEN 状態の DCB を持っているかどうか調べる．OPEN された DCB があり，それが BTAM, QTAM, BISAM, QISAM 以外の場合は，それを CLOSE し，かつその DCB に関連した IOB を IOB 回復待行列から取り除く．

BTAM, QTAM, BISAM, QISAM の DCB があり，かつ OPEN 状態であると ASIR 4 ルーチンに制御権を渡す．そうでないと ASIR 3 ルーチンへ行く．

⑤ ASIR 3 ルーチン

異常終止したタスクのサブタスクは異常終止処理の段階でディスパッチ不能状態にされているのであるが，このルーチンでディスパッチ可能状態に戻される．

RB 待行列上で STAE マクロ命令を出したプログラムに対応する RB に先行する RB の中の旧 PSW の**次の命令アドレス**（next instruction address）が CVT の SVC 3 命令のアドレスをポイントするようにする．これによって，RB をパージすることが可能となる．そのあとで，使用した SCB を SCB 待行列から除去し，STAE マクロ命令を出したプログラムの RB の中の旧 PSW の次の命令アドレス・フィールドに STAE 再試行ルーチンのアドレスを入れる．この結果，STAE 再試行ルーチンの実行が約束されることになる．

ASIR 3 は EXIT マクロ命令を使ってディスパッチャーに制御権を移す．

⑥ ASIR 4 ルーチン

ASIR 2 ルーチンと同様の処理を BTAM, QTAM, BISAM, QISAM の DCB に対して行なう．

以上みてきたような 5 種類の機能が，タスク監視プログラムの TCB に関する間接サービス機能のうちの主要なものである．

3. 監視プログラムに対する内的サービス機能

監視プログラムの内的サービスの主要な機能は，タスク切換えのテストと記録，ユーザー提供のアドレスの妥当性検査，およびタスクの状況変更の3種類である．以下でそれぞれの機能構造について説明してゆくことにする．ただし，ユーザー提供のアドレスの妥当性検査とタスクの状況変更については簡単にふれることにする．

1) タスクの切換えとディスパッチ機能

オペレーティング・システムのもとで複数個のタスクが CPU を競合使用する場合，CPU の使用権については“システム内の作動可能状態のタスクのうち一番優先順位の高いタスクが CPU を使用する”という原則がある．この原則を実行するためには「CPU を使用しているタスクが待ち状態になると自分より優先順位が低いタスクのうち一番優先順位が高い作動可能状態のタスクに CPU の使用権利を譲る」というオペレーション基準が定まる．また，「待ち状態にあったタスクが作動可能状態になると，作動可能状態になったタスクの優先順位と現在 CPU を使用しているタスクの優先順位を比較し，優先順位の高い方のタスクに CPU の使用権利を与える」というオペレーション基準も必要になる．

オペレーティング・システムにおける CPU 使用権の制御は，タスク切換えのテストと記録，およびタスク・ディスパッチングとに分けられる．タスク切換えの必要性のテストは前述のオペレーション基準にもとづいて行なわれ，タスク切換えが必要になるとその旨を特定ブロックに記録する．一方，タスク・ディスパッチングは，タスク切換えが必要なことを記録した特定ブロックをみて CPU を使用するタスクを決定し，そのタスクが CPU を使用できる様にスケジュールする．

① タスク切換えの必要性のテストと記録

タスク切換えの必要性のテストと記録の面倒をみるのが，**タスク切換えルー**

チン（task switching routine）である．タスク切換えルーチンは種々の監視プログラム・ルーチンで利用されるサブルーチンの１つである．このルーチンは，新たに作動可能状態になったタスクが呼び出しタスクに代ってディスパッチできるかどうか判定し，その表示を行なう．

タスク切換えルーチンにブランチするのは，監視プログラムがあるプログラムの要求ブロック（RB）の中の待ちカウント・フィールドをゼロにしたとき，あるいは TCB の中のディスパッチ不能標識をクリアーしたときである．たとえば，POST ルーチンが入出力オペレーションの完了を待ってプログラムを作動可能にしたり，DEQ ルーチンが逐次再使用可能な資源を待っているプログラムを作動可能にしたときにこの種の状態が生じる．いずれの場合にも，監視プログラムでは作動可能となったルーチンがもとのタスクの優先順位よりも高い順位のタスクに属するものであるかどうか，あるいはディスパッチ可能なプログラムとして呼び出しタスクと入れかえるべきかどうかを判断することができない．その結果，監視プログラム・ルーチンはタスク切換えルーチンにブランチするのである．

タスク切換えルーチンは，作動可能となったルーチンが属する TCB の**指名順位**（dispatching priority）と他の TCB の指名順位とを比較する．他の TCB とは，呼び出しタスクの TCB ないしは別の作動可能状態のルーチンが属する TCB である．いずれにしろ，タスク切換えルーチンは比較の結果にもとづいてタスクの切換えの必要性が判明するとその旨の表示を記録にとどめる．その結果，ディスパッチャーはその記録をもとにディスパッチすべきタスクとルーチンを決定するのである．

実際には，タスク切換えの必要性は，中核プログラムの中の IEATCBP から始まる倍ワードの場所に記録される．この場所の最初のワードは新 TCB ポインター，２番目のワードは，旧TCB ポインターと呼ばれる．最初のワードでポイントする TCB と２番目のワードでポイントする TCB が異ったとき，タスク切換えが必要であると判定する仕組みになっている．そこで，新 TCB ポインターに０，または新たに CPU の使用権利を与えたいタスクの TCB アドレスを挿入することによってタスクの切換えの必要性を表示するのである．

新 TCB ポインターを0にするのは，目下 CPU を使用しているタスクが待ち状態に入ったときである．新 TCB ポインターを0にするのは，タスク切換えルーチンの働きではない．タスクを待ち状態にするマクロ命令に対応するシステム・ルーチンの働きである．たとえば，WAIT，ENQ マクロ命令などを出すことによって CPU を使用していたタスクが待ち状態になったとき，それらのマクロ命令の処理ルーチンが新 TCB のポインターを0にする．

一方，タスク切換えルーチンが機能するのは，新たにタスクが作動可能状態になったときである．つまり，先述の CPU 使用権利の2番目のオペレーション基準にもとづいた判定をくだすさいに働くのである．RB の中の待ちカウント・フィールドが0になるか，または TCB の中のディスパッチ不能標識がリセットされると，タスク切換えルーチンに制御権が渡る．タスク切換えルーチンには作動可能になったタスクの TCB アドレスがパラメータとして渡され，つぎの処理が行なわれる．

- パラメータとして渡されたTCBの指名順位よりも新 TCB ポインターでポイントしている TCB の指名順位の方が高いと，タスク切換ルーチンは何もしないで制御権を戻す．
- 逆に，パラメータとして渡されたTCBの指名順位の方が新 TCB ポインターでポイントしている TCB の指名順位より高い場合には，パラメータとして渡された TCB アドレスを新 TCB ポインターに入れる．
- パラメータとして渡されたTCBの指名順位と新 TCB ポインターがポイントする TCB の指名順位が等しいときは，TCB 待行列上の相対的位置が高い方の TCB アドレスを新 TCB ポインターに入れる．この場合，新 TCB ポインターがポイントしている TCB から探索を開始し，パラメータとして渡された TCB がそれより低い位置にあるかどうか判定して相対的位置を決定する．
- 新TCBポインターが0の場合は，新 TCB ポインターの代りに現在 CPU を使用しているタスクの TCB を使用して上記の処理を遂行する．

②　タスクのディスパッチング

上述の処理が終ると，タスク切換えルーチンを呼び出した監視プログラム・ルーチンに制御権が戻される．そのあとで，ディスパッチャーが IEATCBP の新 TCB ポインターと旧 TCB ポインターの内容を比較し実行すべきプログラムを決定する．ディスパッチャーには割込み処理後に制御権が渡り，両者の比較結果をもとにつぎの 3 つの基準にもとづいてタスクのディスパッチングを行なう：

- 両者が等しくなく，かつ新 TCB ポインターが 0 でない時は，新 TCB ポインターが指している TCB に CPU を使用させる．
- 両者が等しくなく，かつ新 TCB ポインターが 0 の時は，目下 CPU を使用している TCB を起点に TCB 待行列を探索し最初にみつかった作動可能状態の TCB に CPU を使用させる．
- 新 TCB ポインターと TCB ポインターが等しい場合にはタスクの切換えは行なわない．

以上のタスクの切換えとディスパッチ機能を具体的な例で示したのが図 8-56 である．これをもとに，まとめておこう．

A のステップでは新旧の TCB ポインターが等しいので，タスクの切換えは生じない．したがって，ディスパッチャーはタスク A に CPU の制御権を戻す．タスク A の実行中に WAIT マクロ命令が出されると，割込みが生じタスク A を待ち状態におくために WAIT ルーチンに行く．

B のステップでは，WAIT ルーチンが CPU を使用しているタスク A を待ち状態におくと同時に新 TCB ポインターを 0 にする．ディスパッチャーは，新 TCB ポインターが 0 なので，タスク A の TCB を起点に TCB 待行列を探索し作動可能状態にあるタスク B に CPU を使用させる．タスク B を遂行中に ENQ マクロ命令が出され，その処理を行なうための割込みが生じる．

C のステップで ENQ ルーチンが機能しタスク B を待ち状態におくと同時に新 TCB ポインターを 0 にする．新 TCB ポインターが 0 になったので，ディスパッチャーはタスク B の TCB を起点に TCB 待行列を探索する．どのタスクも作動可能状態にないのでシステムを一時待ち状態におく．入出力完了など

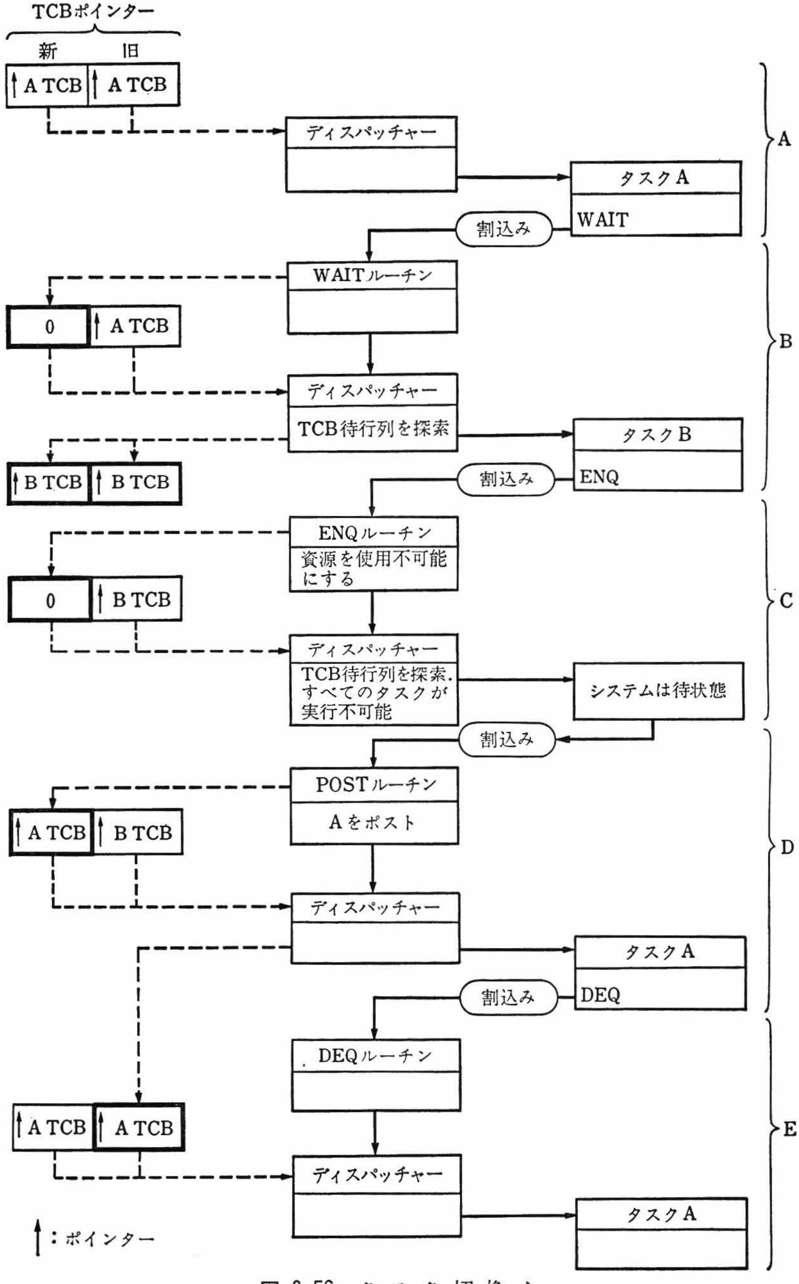

TCBポインター
新
旧
A TCB
A TCB
ディスパッチャー
タスクA
WAIT
割込み
WAITルーチン
0
A TCB
ディスパッチャー
TCB待行列を探索
タスクB
B TCB
B TCB
割込み
ENQ
ENQルーチン
資源を使用不可能にする
0
B TCB
ディスパッチャー
TCB待行列を探索．すべてのタスクが実行不可能
システムは待状態
割込み
POSTルーチン
Aをポスト
A TCB
B TCB
ディスパッチャー
タスクA
割込み
DEQ
DEQルーチン
A TCB
A TCB
ディスパッチャー
タスクA
：ポインター
A
B
C
D
E

図 8-56　タスク切換え

の所要の事象が発生することによって割込みが生じ待ち状態から脱出する．

Dのステップで，POST ルーチンが機能するが，この場合タスク切換えルーチンにブランチし，タスク切換えのテストと記録を行なう．新 TCB ポインターは0であるので，パラメータとして渡されたタスクAの指名順位と目下CPUを使用しているタスクの指名順位を比較することになるが，タスクBは待ち状態に入っているので必然的にタスクAの TCB アドレスを新 TCB ポインターに入れることになる．ディスパッチャーは新旧 TCB ポインターが等しくなく，かつ新 TCB ポインターが0でないことから，新 TCB ポインターがポイントしているタスクAに CPU 制御権を渡す．タスクAの遂行中に DEQ マクロ命令が出され割込みが生じる．

Eのステップでは，まず DEQ ルーチンが機能するがタスク切換えをテストするためにタスク切換ルーチンにブランチする．タスク切換ルーチンは，パラメータとして渡されたタスクBの指名順位の方が，新 TCB ポインターでポイントされているタスクAのそれよりも低いので何らの処理もしない．制御権がディスパッチャーに渡ると，ディスパッチャーは新旧 TCB ポインターを比較するが，両者が等しいのでタスクの切換えは行なわずタスクAに再び制御権を戻す．

このようにして，タスクの切換えが遂行されてゆくのである．

2） ユーザー提供のアドレスの妥当性検査機能

監視プログラム・ルーチンは，**妥当性検査ルーチン**(validity check routine)をサブルーチンとして使用し，ユーザー・プログラムでインプット・パラメータとして受け渡した主記憶装置アドレスをチェックする．妥当性検査ルーチンは，受け渡されたアドレスに関してつぎの諸点の検査をする：

- アドレスが，フルワード境界に合されているかどうか．
- アドレスが主記憶装置のサイズを越えていないかどうか．
- 指定されたアドレスを含む記憶域の記憶保護キーが呼び出しプログラムの TCB で示されている記憶保護キーに一致しているかどうか．

これらのどれかが満足されていないと，検査ルーチンは現 PSW の条件コードを変更することによって監視プログラム・ルーチンにエラーを伝達する．呼び出しプログラムには重大なエラーがあるわけだから，監視プログラム・ルーチンはそのタスクを異常終止させてしまう．この時点でプログラミング上の誤りを検出しておくことによって，あとの処理で生じるプログラム・チェックの割込みを回避しようとしているのである．この種のプログラム・チェックの診断はきわめてむずかしい問題だからだ．そのため，アドレス・エラーの早期発見を目論んでいるのが妥当性検査ルーチンである．

3)　タスクの状況変更機能

監視プログラム・ルーチンは，**状況セット・ルーチン**（set status routine）を使ってタスクの特殊な状況をセットしたり，逆にリセットすることができる．状況セット・ルーチンは，特殊なマクロ命令である STATUS 命令を使って呼び出される．その結果，**SVC 第一次割込み処理ルーチン**（SVC first-level interruption handler）を経由して，このルーチンに制御権が渡る．また，タイプⅠ SVC ルーチンで STATUS マクロ命令が出された場合には直接このルーチンにブランチする（タイプⅠ SVC ルーチンは SVC 割込みを引き起すことができないから）．状況セットの処理が終ると，タイプⅠエグジット・ルーチンを経由してもとのプログラム・ルーチンに制御権が戻される．

ともあれ，状況セット・ルーチンは，特定タスクないしはタスク・グループに対してつぎの3種類の状況をセットしたり，リセットしたりする：

- **ロールアウト不能状況**（nonrolloutable status）：　この結果，特定タスクをロールアウトできない状態にしたり，逆にロールアウトできる状態にすることができる．
- “must complete” status：　現行タスクが遂行されている間，ジョブ・ステップやシステム内の他のタスクをディスパッチできない状態にしたり，逆にディスパッチ可能な状態にすることができる．
- **ディスパッチ不能状況**（nondispatchability status）：　ディスパッチャー

が特定タスクのルーチンを再開することを不能にしたり，逆に再開することを可能にしたりすることができる．

監視プログラムの内的サービス機能は，ユーザー・プログラムで関与することができない機能であり，監視プログラムに対して特別なルーチンがサービスを提供するものである．タスク切換ルーチン，妥当性検査ルーチンおよび状況セット・ルーチンがその代表的なサービス・ルーチンである．

ともあれ，タスク監視プログラムの機能こそは，タスク管理機能の主軸をなすものであり，多重タスキングを実現するために必須のものである．本節では，このタスク監視プログラムの機能を「TCB に関する直接サービス機能」，「TCB に関する間接サービス機能」および「監視プログラムに対する内的サービス機能」に大別して詳述してきたのである．

五．内容監視プログラムの機能構造

要求されたモジュールの所在をつきとめ，必要なら該当モジュールを主記憶装置に呼び込み，その実行をスケジューリングするのが，監視プログラムのうちの内容監視機能である．この目的を遂行するために，主記憶装置あるすべてのプログラム・モジュールの状態を記録，管理する．

内容監視プログラムは，LINK, LOAD, XCTL，あるいは ATTACH マクロ命令が出されたときに CPU の制御権を得る．これらのマクロで要求されたモジュールがどこにあるかどうかを調べ，もし使用可能なモジュールが主記憶装置内に存在しないときには，該当モジュールをライブラリーから捜し出しローディングする．なお，LINK, LOAD, XCTL, ATTACH マクロ命令の機能については，「第四章プログラム設計構造と連係編集プログラム」の「三．プログラム構造と制御プログラムの働き」を参照されたい．

上述したように，内容監視プログラムは主記憶域にあるすべてのプログラム・モジュールに関する記録，つまりモジュールの属性，所在場所あるいは使用状況などの記録を保持している．この記録を一般に**内容登録簿**（contents dir-

バイト			
0	CDATTR：モジュールの属性を示すフィールド	1	CDCHAIN：待行列中のつぎのCDEのアドレス
4	CDROLL：予約フィールド	5	CDRBP：RBのアドレス
8	CDNAME：モジュール名または別名		
16	CDUSE：使用回数または責任回数を示すフィールド	17	CDENTPT：モジュールの入口点アドレス
20	CDATTR2：モジュールの属性を示すフィールド	21	CDXLMJP：エクステント・リスト・アドレス，または該当CDEが副CDEの場合には主CDEのアドレス

図 8-57　内容登録簿記入項目（CDE）の形式と内容

ectory）と呼んでいる．内容登録簿は，**連係パック域制御待行列**（Link Pack Area Control Queue；略して **LPACQ**），**ジョブ・パック域制御待行列**（Job Pack Area Control Queue；略して **JPACQ**），および**ロード・リスト**（load list）の3種類から構成されている（**図 8-59** 参照）．

LPACQ は，連係パック域に収容されているすべてのプログラム・モジュールに関する記録からなる．連係パック域には，言うまでもなく制御プログラムないしはユーザー・プログラムで使用される再入可能プログラムがローディングされている．しかも，ここにあるモジュールは，**NIP**(Nucleus Initialization Program）によってロードされたものであり常駐プログラムである．ともあれ，LPACQ は**内容登録簿記入項目**（Contents Directory Entry; 略して **CDE**）

バイト0	1	4	5
0	LLCHAIN：ロード・リスト上のつぎの LLE の最初のバイトを示すアドレス	LLCOUNT：責任回数	LLCDPTR：該当モジュールに対応するCDEのアドレス

図 8-58　ロード・リスト要素（LLE）の形式と内容

から成っている．CDE は待行列上の実際のプログラム・モジュール を表わしている．つまり，CDE はモジュールごとに作られ，そのモジュールの状態，名前，入口点アドレスなどの情報を保持している．この形式と内容を示したのが**図 8-57** である．モジュールの状態は，CDATTR と CDATTR 2 の 2 つのフィールドで示される．モジュール名は CDNAME フィールドに，また入口点アドレスは CDENTPT フィールドに収容される．

あるタスクで LPACQ の中で示されているモジュールを 要求すると，そのモジュールは該当タスクの RB 待行列の中で PRB を使って示される仕組みになっている．この PRB のアドレスが CDE の CDRBP フィールドに挿入されることによって両者が関係づけられるのである．

LPACQ は，システム待行列域に作られ，コミュニケーション・ベクトル・テーブルからポイントされる（**図 8-59** ①参照）．CDE は CDCHAIN フィールドを使って連鎖される．また，CDE は**エクステント・リスト**（extent list；略して **XL**）を指している．この XL で主記憶装置にローディングされているモジュールの場所を示しているのである．さらに，XL にはモジュールによって占められている記憶域の大きさだとか，モジュールを構成する制御セクション（CSECT）の大きさや入口点などが保持されている．

一方，JPACQ は各ジョブ・パック域ごとに作られる．ジョブ・パック域は各領域ごとに作られ，連係パック域にないモジュールを保持するのに用いられる．JPACQ は，LPACQ と同様に CDE から構成されている．JPACQ の CDE は，ジョブ・ステップ内のタスクを遂行するために内容監視プログラムが主記憶装置に呼び込んだジョブ・ステップ領域のルーチンに関する情報を持っている．ジョブ・パック域にローディングされたルーチンは再入可能なものであってもよいし，そうでなくてもよい．いずれにしろ，連係パック域に収容されているモジュールと異なり，そのジョブ・ステップ内のタスクを遂行するためにしか使用することができない．

JPACQ の様子を示したのが，**図 8-59** の②である．LPACQ が CVT でポイントされているのに対して，JPACQ はジョブ・ステップ用の TCB からポイントされる．LPACQ はシステム内の すべての タスクに 対して 適用される

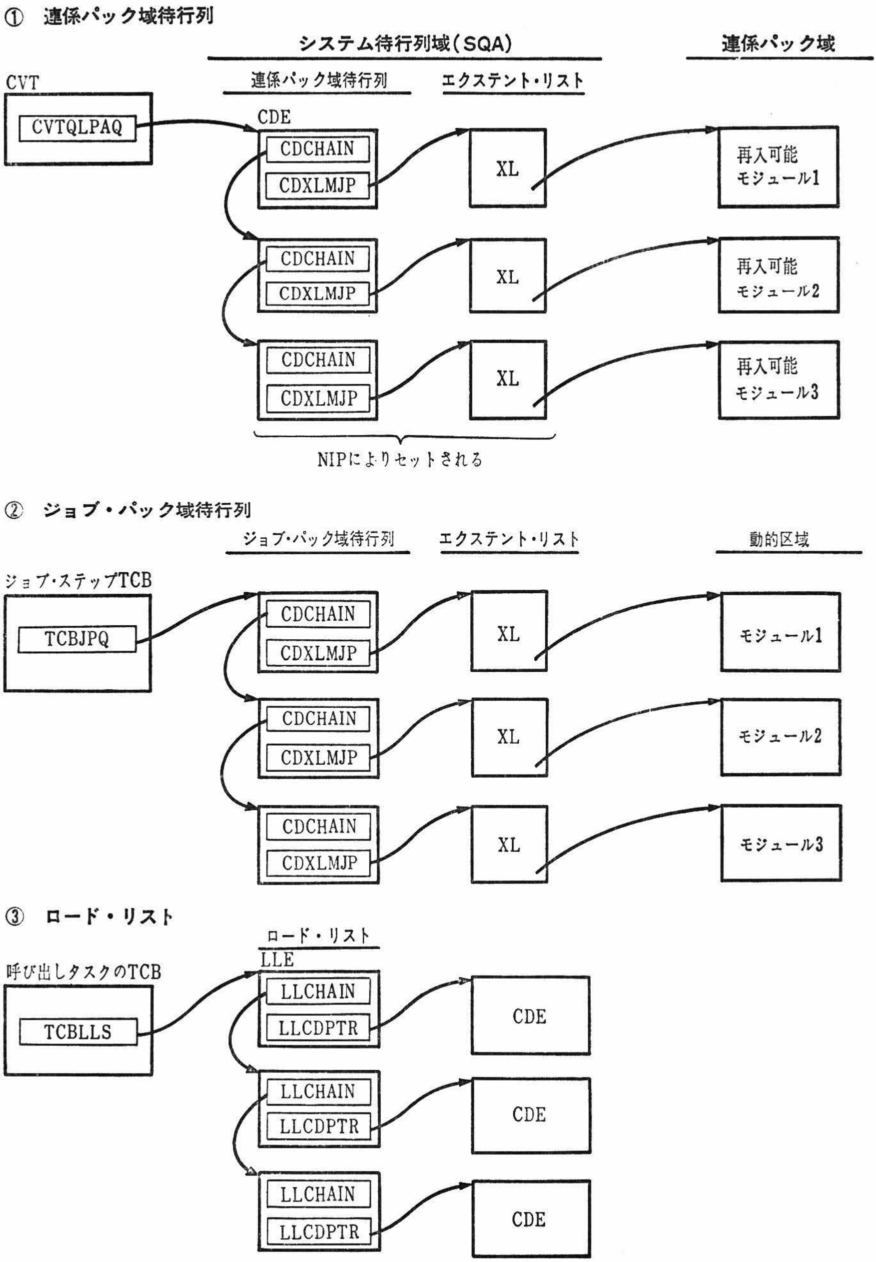


図 8-59　内容登録簿 (CD)

が，JPACQ は該当ジョブ・ステップに対してだけ意味を持っているからである．JPACQ 内の CDE は，JPACQ の場合と同様に XL をポイントしている．また，CDCHAIN フィールドを用いて，CDE 相互間の連鎖を保っている．

ロード・リストは，内容監視プログラムがジョブ・パック域に呼び込んだ，または連係パック域で見つけ出したルーチンで，かつ LOAD マクロ命令の機能を果たすルーチンを表示している．ロード・リストのエントリーは，CDE ではなくて，LOAD マクロ命令によって作られる**ロード・リスト要素**（Load List Element；略して **LLE**）で構成される．**図 8-58** で示しているように，LLE は該当モジュールに対する LOAD 要求の回数（責任回数）とロード・リスト上の連鎖関係，および JPACQ ないし LPACQ の中の CDE との関係づけを行なっている．したがって，ロード・リスト内で示されているプログラム・モジュールは別の内容登録簿待行列の構成要素の１つをも表示しているのである．これを具体的に示したのが，**図 8-59** の③と**図 8-60** とである．

ロード・リストは，LOAD マクロ命令を出したタスクごとに作成され，そのタスクの TCB でポイントされている．ロード・リスト内の各要素（LLE）は，LLCHAIN フィールドを用いて相互に連鎖されていると同時に，LLCDPTR フィールドで該当モジュールに対応する CDE を指示している．図 8-60 の例では，ジョブ・パック域にローディングされたモジュールしかポイントしていないが，場合によっては連係パック域に常駐している再入可能モジュールをポイントすることもある．

これまで説明してきた内容登録簿と他の関連する主要な制御ブロックとをまとめると**図 8-61** のようになる．LPACQ は，CVT から直接ポイントされる．また，ロード・リスト内の LLE から LPACQ 内の特定 CDE がポイントされることもある．一方，JPACQ 自体はジョブ・ステップ用 TCB からポイントされ，待行列内の特定 CDE は該当プログラム・モジュールを表わす RB からポイントされる．LPACQ の場合と同じように，LOAD モジュールであるとその CDE はロード・リスト内の LLE からポイントされる．なお，LPACQ にしろ JPACQ にしろ，その中の各々の CBE は必ず XL を指示している．XL は，実際にローディングされているモジュールの大きさやその所在を表示

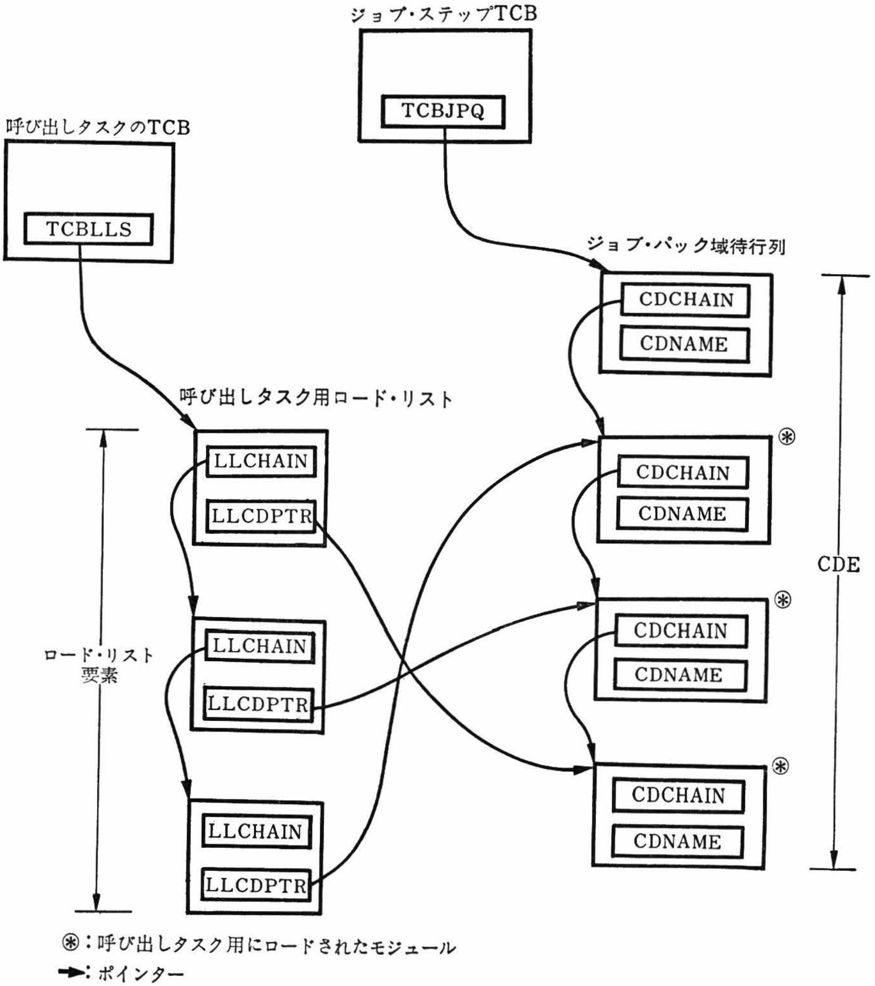


図 8-60　ロード・リストとジョブ・パック域待行列の相互関係

しているからである.

内容監視プログラムの処理は，TCB 待行列と非常に密接な関係をもった待行列である要求ブロック (RB) を通じて遂行される (図 8-61 参照). したがって，ここで，RB に関して若干まとめておこう.

RB は，あるタスク内での制御のレベルを表わしている. 内容監視プログラ

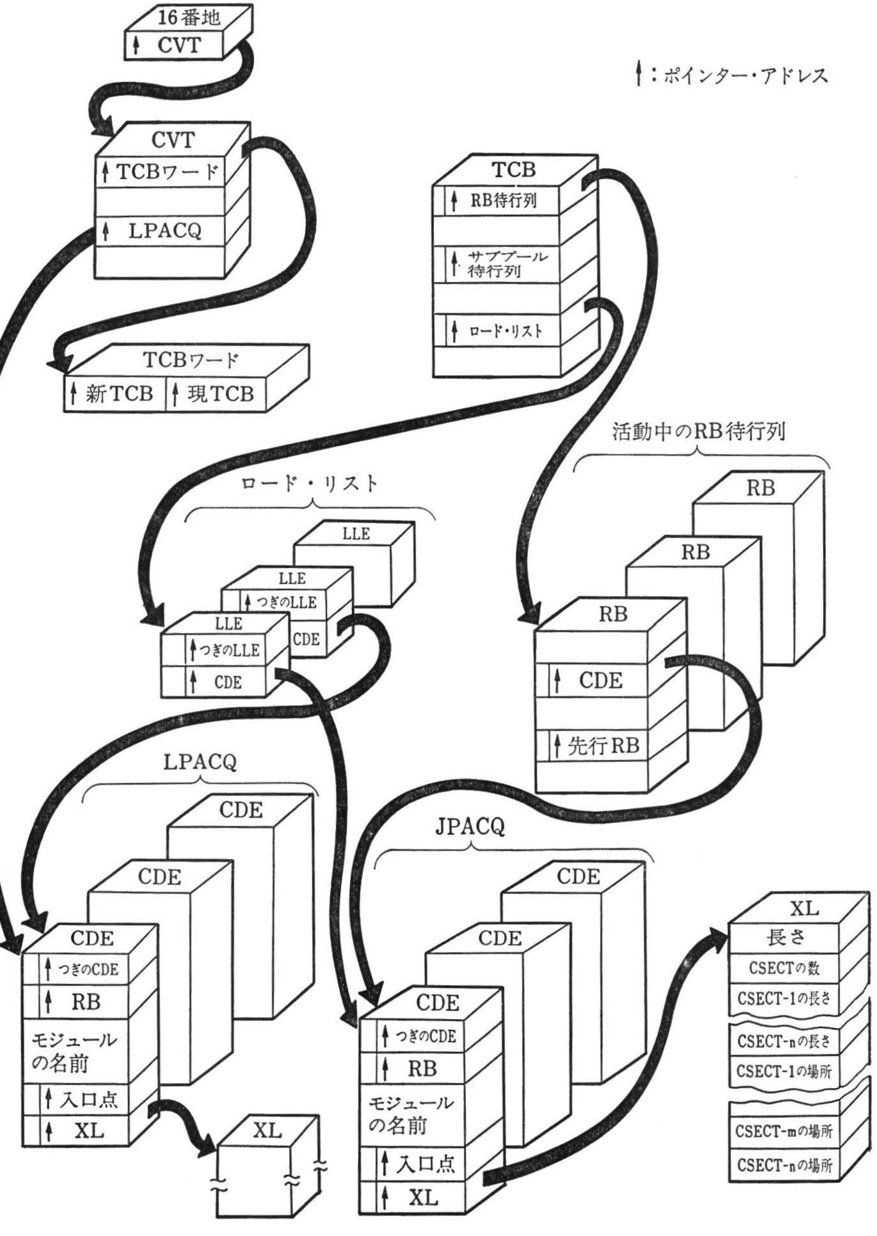


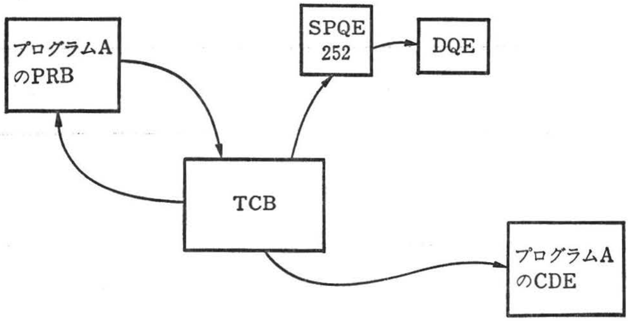
図 8-61　制御ブロックの相互関係

ムは，新規のタスクを遂行するのに最初の制御レベル用の RB を作りあげる．この RB および後続レベルの RB が TCB の RB 待行列上で連鎖される．たとえば，**図 8-62** をみて欲しい．プログラム A が最初に制御権を得たプログラムである．このプログラムAが，プログラムBを指定した LINK マクロ命令を出すと，プログラムBの制御レベルを表わす新規の RB を作成し，RB 待行列に追加され，プログラムBの PRB は RB 待行列のトップに置かれる．タスクが制御機能を得たとき実際に実行されるのは，そのタスクの RB 待行列のトップにある RB で表示されているルーチンである．したがって，**図 8-62** の **2.** の状態では，プログラムBが実行される．プログラムBの処理が完了すると，制御権はプログラムAに戻される．さらに，プログラムCを LINK するとプログラムBの場合と同様の処理がなされる．いずれにしろ，RB が作られるごとにそれに対応する CDE が作成され内容登録簿に記入される．ともあれ，いくつかの RB が待行列に入っている時に，2番目の RB で表示されているルーチンが制御権を得るのは，トップの RB がなくなったとき，すなわちトップの RB で示されているルーチンの処理が終了し，そのルーチンから制御権が戻されたときである．

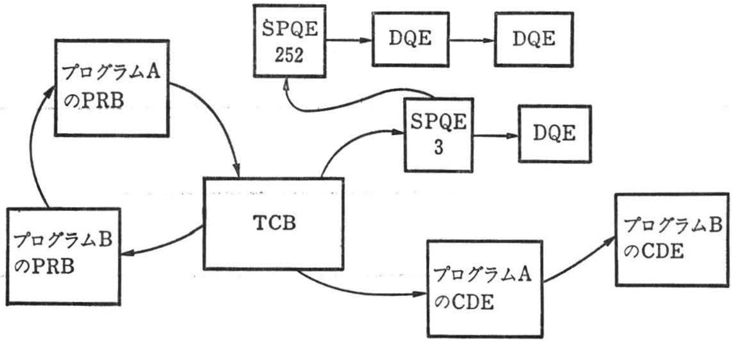
ところで，RB には4種類のタイプがある．**プログラム要求ブロック**（Program Request Block；略して **PRB**），**割込み要求ブロック**（Interruption Request Block；略して **IRB**），**監視プログラム要求ブロック**（Supervisor Request Block；略して **SVRB**）および**システム割込み要求ブロック**（System Interruption Request Block；略して **SIRB**）の4種類である：

- PRB：　タスクを遂行するのに必要なプログラムに関する情報を保持するのに用いられる．PRB はタスクを実行するさいの非監視プログラム・ルーチンを表示する．PRB は，ATTACH, LINK, XCTL 機能を遂行する内容監視プログラム・ルーチンで作られる（PRB の形式と内容に関しては図 8-38 を参照のこと）．
- IRB：　非同期的に実行されてルーチンに関する情報を保持するのに使われるのが IRB である．つまり，IRB は非同期的な割込み事象が生じたときに実行するルーチンを制御する．IRB は割込みが生じる前にユーザー・

1. 最初の状態



2. プログラムBをLINKすると



3. プログラムBの処理が終り，さらにプログラムCをLINKすると

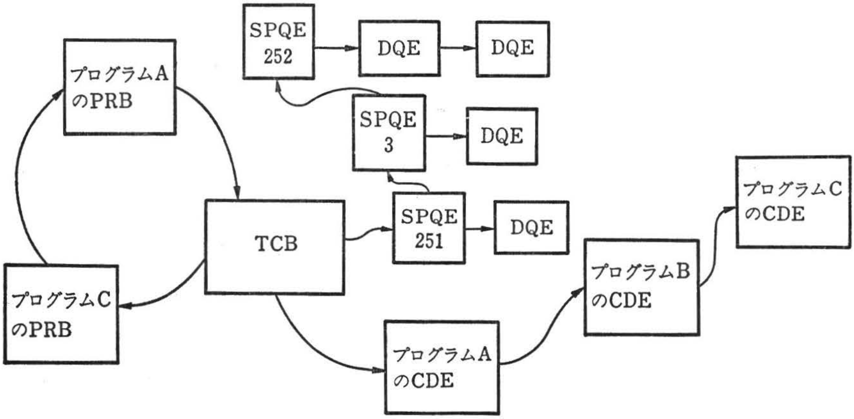


図 8-62 CDE と RB との関係

バイト			
0	予　　約		
4	RBABOPSW：ABTERMルーチン実行後のユーザー旧PSWの右半分，その他の場合はゼロ		
8	RBWCSA：終了時の待ち要求回数（待ちカウント保存域）	9 RBSIZE：RBのサイズ	10 RBSTAB：状況ビットおよび属性ビット
12	RBCDFLGS：内容制御標識	13 RBCDE：該当要求ブロックが関連するCDEのアドレス	
16	RBOPSW：ユーザー旧PSW		
24	ゼ　ロ	25 RBPGMQ：同一逐次再使用可能プログラムの使用要求を表示しているRBのアドレス	
28	RBWCF：待ち要求回数(待ちカウント)	29 RBLINK：先行RBのアドレス，またはTCBのアドレス	
32	RBGRSAVE：監視プログラムで使用する汎用レジスター保存域		
96	RBEXSAVE：SVCルーチン用の拡張保存域　143		

図 8-63　監視プログラム要求ブロック（SVRB）——常駐 SVC ルーチン用

プログラムからの要請にもとづいて CIRB ルーチンで作成される．しかし，実際に割込みが生じてるまで RB 待行列には入れられない．これに関する詳細は，「ユーザ・エグジット・ルーチンのスケジューリング機能」の個所でふれている（IRB の形式と内容に関しては図 8-50 を参照）．

● SVRB：　タイプⅡ，Ⅲ，Ⅳの SVC ルーチンに関する情報を保持するのが SVRB である．SVRB は SVC 割込み処理ルーチンで作成され，PRB と同じようにタスクの RB 待行列に挿入される．

SVRB には，2つの形式がある．1つは，タイプⅡの SVC ルーチンに関するもので，常駐 SVC ルーチンに対応した SVRB である．この形式と内容とを示したのが**図 8-63** である．PRB の形式と基本的には変えら

バイト			
0	RBTABNO：一時域制御ブロック(TACT)の始点から当該SVRBで表示されるモジュール・エントリまでの変位	2 RBRTLNTH：SVCルーチンの長さ(バイト数)	
4	RBABOPSW：一時域処理ルーチン実行後の要求ルーチン名の下位4バイト．ABTERMルーチン実行後はユーザー旧PSWの右半分		
8	RBWCSA：終了時の待ち要求回数(待ちカウント保存域)	9 RBSIZE：RBのサイズ	10 RBSTAB：状況ビットおよび属性ビット
12	RBSVTQN：一時ルーチン待行列上のつぎのRBのアドレス		
16	RBOPSW：ユーザー旧PSW		
24	RBTAWSCA：待ちカウント・オーバレー保存域	25 RBSVTTR：SVCルーチンのTTR形式のアドレス	
28	RBWCF：待ち要求回数(待ちカウント)	29 RBLINK：先行RBのアドレス，またはTCBのアドレス	
32	RBGRSAVE：監視プログラムで使用する汎用レジスター保存域		
96	RBEXSAVE：SVCルーチン用の拡張保存域		
143			

図 8-64 監視プログラム要求ブロック (SVRB)——一時 SVC ルーチン用

れないが，監視プログラム・ルーチンで使用する汎用レジスター保存域と拡張保存域が設けられている点が異なる．もう1つの形式は，タイプⅢとⅣの SVC ルーチンに関するもので，一時 SVC ルーチンに対応した SVRB である．この形式と内容は図 8-64 で示している．常駐 SVC ルーチン用の SVRB にいくつかの新しいフィールドが追加されている．

● SIRB： 入出力エラー処理ルーチンに関する情報を保持するのが SIRB である．SIRB はシステム・エラー用タスクに対してのみ使用され，しかもシステムに唯1つの SIRB しか存在しない．SIRB の形式と内容は図 8-65 で示している．他の RB と異なり，入出力エラー・ルーチン名だとか入口点のアドレスが収容されている．また，CDE をポイントするアドレスを

バイト			
0	RBEXRTNM：入出力エラーのエグジット・ルーチン名		
8	RBWCSA：待ちカウントの保存域	9 RBSIZE：該当RBのサイズ	10 RBSTAB：状況ビットと属性ビット
12	RBEP：非同期的に実行するルーチンの入口点アドレス		
16	RBOPSW：ユーザーの旧PSW		
24	予約フィールド		RBIQE：要求待行列要素(IQE)のためのリスト起点
28	RBWCF：待ち要求回数（待ちカウント）	29 RBLINK：先行RBのアドレス，またはTCBのアドレス	
32	RBGRSAVE：汎用レジスター保存域		91

図 8-65　システム割込み要求ブロック（SIRB）の形式と内容

収容するフィールドは存在しない．

以上4種類の RB うち，内容監視プログラムは PRB だけを作りあげる．いずれの RB も TCB がポイントする RB 待行列に入れられ，プログラムの実行を制御するのに用いられる．この様子をまとめたのが，**図 8-66** である．

1.　内容監視プログラムの機能構成

内容監視プログラムには，大別して**共通機能**（common function）と**特殊機能**（special function）との2つのタイプで構成されている．共通機能は，プログラム・モジュールの連係要求，または将来の使用にそなえてモジュールを主記憶域にフェッチする要求にこたえる．この共通機能は，LINK, LOAD, XCTL および ATTACH マクロ命令を使って要求される．しかも，**共通サブルーチ**

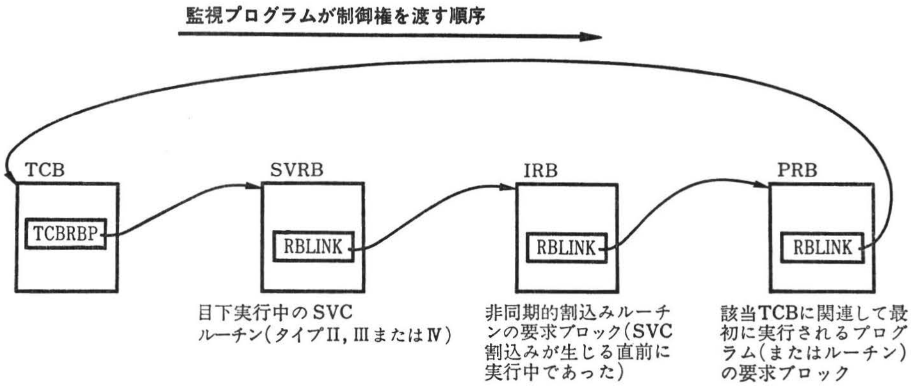


図 8-66　要求ブロック (RB) 待行列

ン (common subroutine) と呼ばれる一群のサブルーチンでこれら機能が遂行される．一方，特殊機能はシステムないしはユーザ・ルーチンからの特別な要求にこたえるとか，共通機能の援助的な働きをする．たとえば，モジュールの入口点の表示とか，オーバレイ構造のセグメントのローディングとかが特殊機能に相当する．

共通機能は，つぎの8種類の機能から構成される：

1）要求モジュールに応じて内容登録簿を探索する．

2）必要ならば要求モジュールを表示する CDE を作成し，要求パラメータにもとづいて CDE に所定の情報を入れる．さらに，作りあげた CDE を対応する内容登録簿待行列に挿入する．

3）モジュールの状態をテストし，使用可能かどうか判定する．モジュール状態がテストされるのは，CDE が内容登録簿待行列に見出されたとき，ないしはモジュールに対して BLDL 手順がとられたときである．

4）主記憶域に存在しないモジュール，またはあっても再使用可能でないモジュールをフェッチする．

5）**別名** (alias name) でモジュールが要求された場合には，別名の入口点を

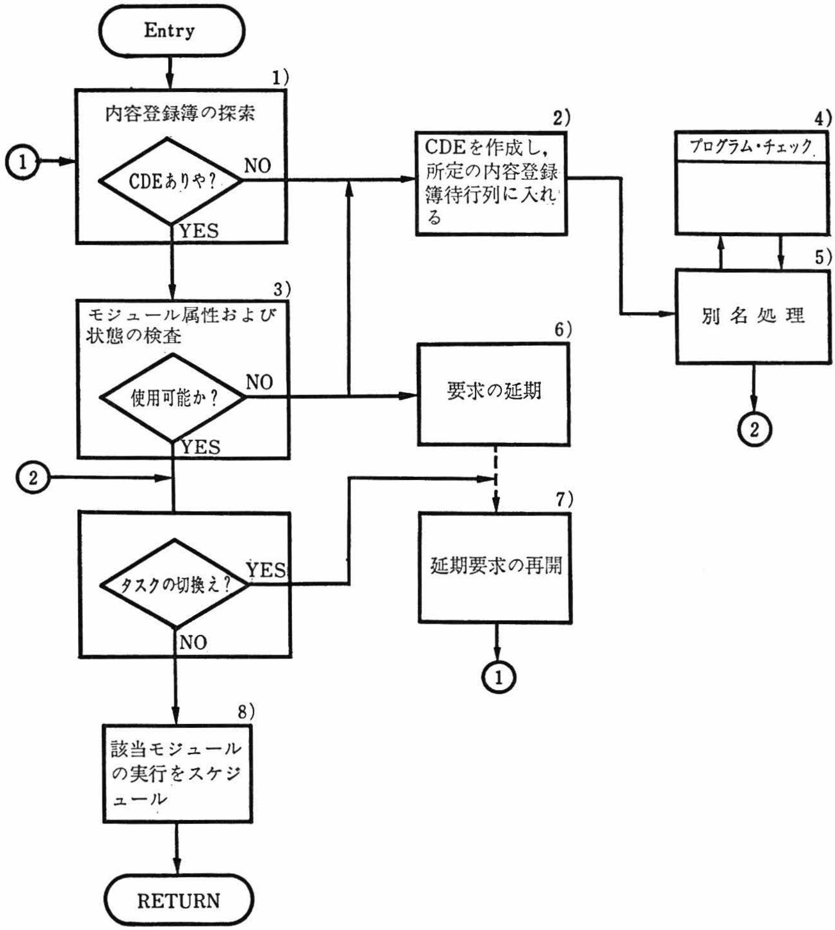


図 8-67　共通機能の流れ

決定し，対応する内容登録待行列を更新する．

6）該当モジュールが即時使用可能でない場合には，その要求を延期する．

7）該当モジュールが使用可能となった時点で，延期されていた要求の処理を再開する．

8）PRB を作成し，それを呼び出しタスクの RB 待行列上にある SVRB の直後に入れることによって，モジュールの実行をスケジュールする．

以上の８種類の共通機能を，制御の流れとして表現すると**図 8-67** で示したようになる．各々のブロックの右上に示した番号が上述の機能の先頭に附加した番号に対応する．それぞれの詳細は後述する．

特殊機能は，共通機能のいずれかの補助，または特殊なサービスを行なう．特殊処理は，LOAD 要求および SVC ルーチンで出す XCTL 要求にもとづいて遂行される．一方，共通機能の補助としては IDENTIFY 要求を処理することによって，指定されたモジュール内の入口点を監視プログラムに伝える．また，DELETE 要求を処理して，LOAD 要求でローディングされたモジュールがもはや必要なくなったことを監視プログラムに知らせる．モジュールをオーバレイ構造でローディングしなければならないときには，オーバレイ監視プログラムが呼び出され，所定のセグメントのローディングのための前準備を行ない，ローディングの制御をする．この場合，実際のローディングは，プログラム・フェッチ・ルーチンで処理される．このルーチンは，内容監視プログラム，一時域フェッチ・ルーチン，オーバレイ監視プログラムおよびステージ・エグジット・エフェクターに対するローダーとして機能する．

特殊機能の主要なものとしては，以上の６種類，つまり LOAD 要求に対する処理，XCTL の処理，IDENTIFY の処理，DELETE の処理，オーバレイ監視機能およびプログラム・フェッチである．

2.　共通機能の働き

LINK, LOAD, または ATTACH 要求が出されると，SVC SLIH を経由して共通サブルーチンに制御権が渡される．内容監視プログラムは，モジュール要求のタイプに応じた特別な初期値化と入力処理を行なう．そのあとで，共通サブルーチンが要求にこたえたサービスを行ない，要求モジュールの所在のつきとめ，モジュール状態の判定，モジュールが利用可能かどうかのテストを行なう．該当モジュールが主記憶域にあり，再入可能であるかまたは使用中でない逐次再使用可能モジュールであると利用可能なモジュールとみなされる．また，再使用不可能なモジュールであっても，まだ一度も使用されていなければやはり利用可能なモジュールである．いずれにしろ，モジュールが使用可能で

あれば，その実行がスケジュールされる．一方，利用可能でない時には，該当モジュールが補助記憶装置からフェッチされてから，実行がスケジュールされる．しかし，モジュールのフェッチができないとすると，その要求は一時延期されざるを得ない．

1）　モジュールの探索機能

内容監視プログラムの第1の機能は，要求されたモジュールがどこにあるか捜すことである．該当モジュールはいくつかの場所に存在し得る．主記憶域のジョブ・ステップ領域，補助記憶装置上の各種ライブラリー，あるいは連係パック域などが考えられる．内容監視プログラムは，まずジョブ・ステップ領域を捜す．そのあとで，補助記憶装置上のライブラリーを捜し，さらに連係パック域を探索する．これらのいずれの場所にも該当モジュールが存在しないとなると最後にリンク・ライブラリーを捜す．

モジュールの探索は，上述のようにジョブ・ステップ領域から開始される．領域では，ジョブ・パック域と呼ばれるサブプールにモジュールを割り当てる．すでに指摘したように，各ジョブ・ステップは独自の JPACQ をもっている．JPACQ には，その領域のジョブ・パック域に存在するモジュールを示す CDE が収容されている．したがって，ジョブ・ステップ領域そのものを探索する代りに JPACQ を構成する CDE を調べることによって，要求モジュールが存在するかどうかを判定する．具体的には，要求されたモジュール名とCDE の CDNAME フィールドに貯えられているモジュール名とを比較してゆくことによって，モジュールの探索が遂行されて行く．

なお，LOAD 要求に対しては，JPACQ を試べるのに先立って，要求したタスクのロード・リストを捜す．ロード・リストには，すでに説明したように LOAD マクロ命令でローディングされるモジュールに対応する CDE をポイントしたフィールドを構成要素としている LLE が収容されている．したがって，ロード・リストからそのモジュールに対応する CDE を見つけ出すのである．

該当ロード・モジュールに対する CDE が見つかると，そのモジュールの状

態（直ちに使用可能か，直ちに使用不可能か，全く使用不可能か）を調べる．一方，該当モジュールの CDE がみつからなかった場合には，そのモジュールがジョブ・パック域に存在しないことを意味するので，所定のライブラリーの探索に移ることになる．

どのライブラリーを捜すかは，モジュールを要求したマクロ命令の DCB オペランドが指定されているかどうかによる．DCB オペランドが指定されていると，指定されているライブラリーを捜す．ライブラリーを捜すにあたっては，BLDL ルーチンを用い，指定されたライブラリーの登録簿を読み込み，この登録簿を調べることによって要求されたモジュールがあるかどうか判定する，一方，DCB＝0 または DCB オペランドが省略されていると，タスク・ライブラリー，ジョブ・ライブラリーの順で探索される．DCB オペランドで 0 が指定されているとか，このオペランドが省略されていることは，特定ライブラリーが指定されていないことを意味するので，タスク・ライブラリー（もしあれば）の登録簿を読み込んで調べる．タスク・ライブラリーが指示されていないとか，タスク・ライブラリーに該当モジュールが存在しないことが分ると，上位のタスク（つまり，親タスク）のタスク・ライブラリーを捜す．この操作はモジュールがみつからないと，ジョブ・ステップ・タスクに到達するまで続けられる．最後には，ジョブ・ステップ・タスク用のライブラリーであるジョブ・ライブラリーの登録簿が調べられるのである．

ここまで探索されても，要求されているモジュールが見つからないと（実際には，モジュールの入口点名が見つからないと），もう一つの内容登録簿待行列である LPACQ を調べる．この探索は，ジョブ・パック域を捜した時と全く同様の方法で行なわれる．異なるのは，JPACQ ではなく LPACQ の CDE を用いた探索である点だけである．

LPACQ を捜しても要求モジュール名がみつからないと，最後にはリンク・ライブラリーを調べる．これは，上述のライブラリー探索と同様に BLDL ルーチンを用いて行なわれる．リンク・ライブラリーの登録簿にも該当ジュールの入口点名が発見されないとすると，呼び出しタスクで誤った要求を出したものと想定される．したがって，最終的には ABEND ルーチンに制御権を渡し，

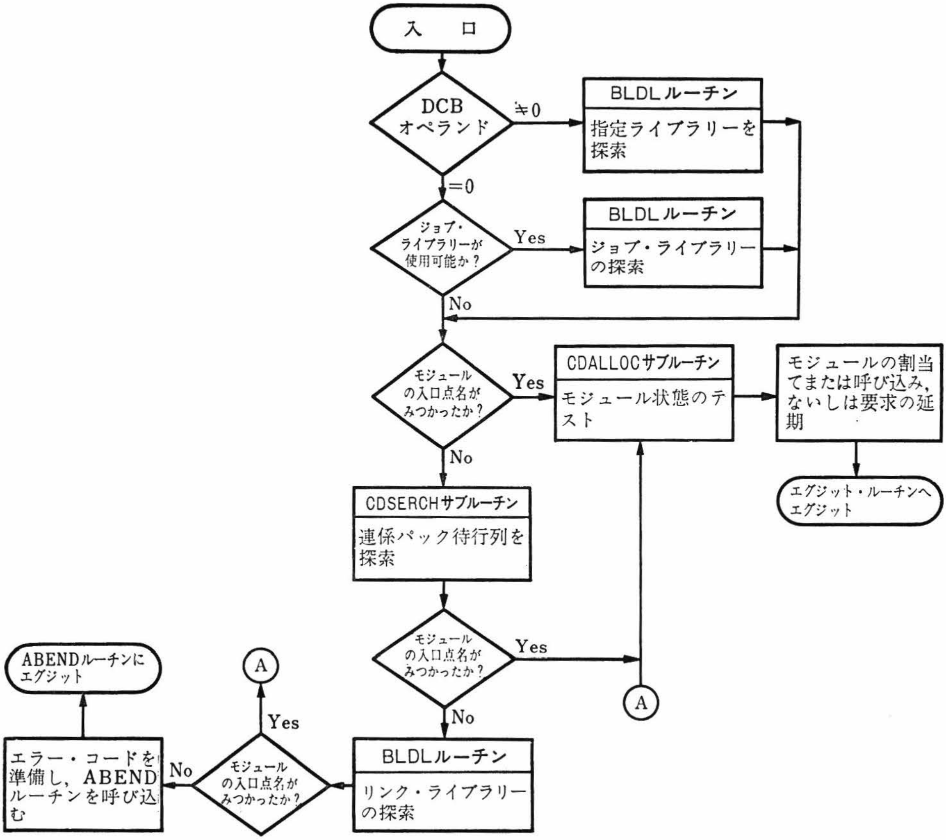


図 8-68　内容監視プログラムを構成する共通サブルーチンの探索（ジョブ・パック域制御待行列にも該当モジュールのCDEがない場合）

そのタスクを異常終止させることになる．

JPACQ に要求モジュールの CDE がない場合には，補助記憶装置上のライブラリーを探索する作業に入るが，これ以後の共通サブルーチンの探索の様子をまとめたのが**図 8-68**である．この図をもとにモジュール探索の手順を各自で整理して欲しい．

2）CDE の作成機能

JPACQ に要求モジュールの CDE がない場合，ライブラリーを捜すに先立

って CDE を収容する場所を GETMAIN マクロ命令で確保してから，そのモジュールのための CDE を作成して JPACQ にチェインする．この場合，CDE の初期値化作業としてつぎのことを行なう：

- 属性フィールド(CDATTR フィールドと CDATTR 2 フィールドのすべてのビットをオンにセットする．
- 入口点アドレス・フィールド（CDENTP フィールド）とエクステント・アドレス・フィールド（CDXLMJP フィールド）をゼロにセットする．

なお，ライブラリーの登録簿を捜した結果，要求モジュールが存在しなかった場合には，新規に作成し挿入した CDE を JPACQ から取り除く．

3) モジュール状態のテスト機能

要求モジュールに対応する CDE が JPACQ か LPACQ にあった場合には，CDE の中の属性フィールドを調べ，そのモジュールが直ちに使用可能かどうかテストする．テストの結果，次の 3 つの状態が判定される：

- 該当モジュールは直ちに使用可能
- 該当モジュールは直ちに使用できないので，要求は待行列におかれる(要求の延期)
- 該当モジュールは使用できないので，ライブラリーから新しいコピーをフエッチしてくる

要求モジュールが直ちに使用可能であると，モジュールに対する未処理の要求回数が記録されている使用回数または責任回数を示すフィールド（CDUSE フィールド）を 1 だけ増やして，要求タスクにそのモジュールの割り当てを行なう．“割り当て”という表現は，実は要求タイプによって異った事柄を意味している．つまり，LOAD 要求に対しては“割り当て”とは単に要求に対する LLE が存在していることを確認することを意味するにすぎない．LLE には，該当モジュールに対する未処理の LOAD 要求の回数を示す責任回数フィールド（LLCOUNT フィールド）と該当モジュールを記述している CDE をポ

イントするフィールド(LLCDPTR) とがある．したがって，この存在を確認することが“割り当て”に相当するのである．一方，LOAD 要求以外の要求タイプでは，モジュールの実行を制御する PRB を作成して，呼び出しタスクの RB 待行列上に挿入し，PRB の各フィールドを初期値化するのが“割り当て”ということになる．つまり，この場合には，モジュールの実行をスケジューリングすることである．

4） プログラム・モジュールのフェッチ機能

要求モジュールが連係パック域にもジョブ・パック域にも存在しないとか，存在していても再入不可能なルーチンでかつすでに使用ずみである場合には，新しいコピーを補助記憶装置のライブラリーからフェッチしなければならない．すでにふれたように，ライブラリーの登録簿は BLDL ルーチンで読み込まれ，該当モジュールがあるかどうか調べられる．該当モジュールがつきとめられると，そのモジュールはサブプール 251 または 252 へフェッチされる．該当モジュールが再入可能であり，リンク・ライブラリーか SVC ライブラリーにあるときにはサブプール 252 にフェッチされる．これ以外のモジュールだとサブプール 251 に収容される．

モジュールのフェッチが完了すると，前に作成した CDE を下記のように更新する：

- フェッチされたモジュールの入口点アドレスを CDENTPT フィールドにセットする．
- **登録簿記入項目** (directory entry) から得たモジュール属性を，属性フィールドにセットする．
- **NIS** (not in storage) ビットをクリアする．
- モジュールがリフレシャブルである場合には，CDATTR 2 フィールドの該当ビットをオンにする．

5） 別名の処理機能

モジュールが別名を使って要求された場合には，特別な処理が行なわれる．つまり，BLDL ルーチンで登録簿の記入項目が得られ，要求されたモジュール名が別名であることが判ると，本来のメンバー名を用いて JPACQ を再び捜す．そのメンバー名に対する CDE があると，その CDE を取り除き，新たに別名のための副 CDE を作り JPACQ にチェインする．副 CDE は，CDNAME フィールドに別名を収容し，CDXLMJP には主 CDE のアドレスを収容する．別名に対応する CDE だけでなく，そのメンバー名に対する主 CDE も同時に作りあげ，やはり JPACQ にチェインする．主 CDE には当然のことながら本名，つまりメンバー名が収容されており，副 CDE からポイントされている．

6） 要求の延期機能

該当モジュールが使用不可能な場合には，要求タスクに該当モジュールをすぐに割り当てることができないので待ち状態におかれる．モジュールに対する要求が延期されるのは，先行する要求で目下該当モジュールがフェッチされている最中である場合，または該当モジュールが主記憶装置内にあるが逐次再使用モジュールでしかも現在使用中である場合である．いずれにしろ，制御権を得ていた SVRB が，待ち状態の SVRB のリストに入れられてしまう．この SVRB リストはサービスすることのできないモジュール要求を示している．この結果，目下のモジュール要求を続行することはできなくなり，タスクの切換えの必要性をディスパッチャーに知らせることになる．

7） 延期要求の再開機能

該当モジュールが使用可能状態になったら，そのモジュールに対する要求は再開始される．再開始のための準備は，BLDL ルーチンがライブラリーの登録簿記入項目を見つけたあと，およびプログラム・フェッチ・ルーチンがモジュールをフェッチしたあとで，内容監視プログラムが行なう．また，逐次再使用可能モジュールに対する要求の再開始は，目下そのモジュールを使用している PRB が取り除かれたあと，エグジット・ルーチンで準備される．

SVRB を待ち状態の SVRB リストから取り除いてビットをクリアし，RB

の旧 PSW に内容監視プログラムの探索ルーチンのアドレスをセットする．そのあと，タスク切換えが必要かどうか調べ，必要ならばタスク切換えを行なう．

8）モジュールの実行のスケジューリング機能

該当モジュールが直ちに使用可能であると，CDE の中の属性フィールド（CDATTR 2）の "release" 標識をゼロにし，使用／責任回数フィールド（CDUSE フィールド）に1を加える．"release" 標識をオフにすることによって，GETMAIN ルーチンに対しそのモジュールのために使われている記憶域は他の目的で作用できない旨を知らせる．一方，使用／責任回数フィールドは，そのモジュールに対する使用要求（LINK, LOAD, XCTL, ATTACH で要求された）が何回残っているかを示す．したがって，このモジュールの使用要求を受けたことによって1を加えるのである．なお，このフィールドはモジュールの使用が終ったとき DELETE ルーチンまたはエグジット・ルーチンにより1ずつ引かれる．

このあと，延期要求に対してチャンスを与えるためにタスク切換えが必要であるかどうか調べる．タスク切換えが必要なければ，そのモジュールに対応する PRB のための記憶域を確保し，PRB を作成して初期値化する．この場合，PRB の情報は，現行 SVRB から得る．また，PRB の RBOPSW フィールドには CDE に収容されているモジュールの入口点アドレスをセットする．

内容監視プログラムは，新規に作成した PRB を RB 待行列の中の SVRB（これは，内容監視プログラムが使用しているのだが）の直後に挿入し，エグジット・ルーチンに制御権を受渡す．これで，モジュールの実行スケジューリングが完了するのである．

以上が内容監視プログラムの共通機能の働きである．

．特殊機能の働き

特殊機能は，すでに説明したような共通機能の補助的役割りと要求タスクに対する特別なサービス機能を果す．その機能はつぎの6種類に要約できる：

- LOAD 要求に対する最終的処理機能
- XCTL 要求に対する特殊な処理機能
- 監視プログラムにモジュールの入口点を知らせる機能（IDENTIFY機能）
- 監視プログラムに LOAD マクロ命令でフェッチしたモジュールが不必要になった旨を知らせる機能（DELETE 機能）
- オーバレイ・モジュールを構成するセグメントのローディングを監視する機能
- モジュールを主記憶装置にフェッチする機能

以上の 6 種類の機能を説明して行くことにする．

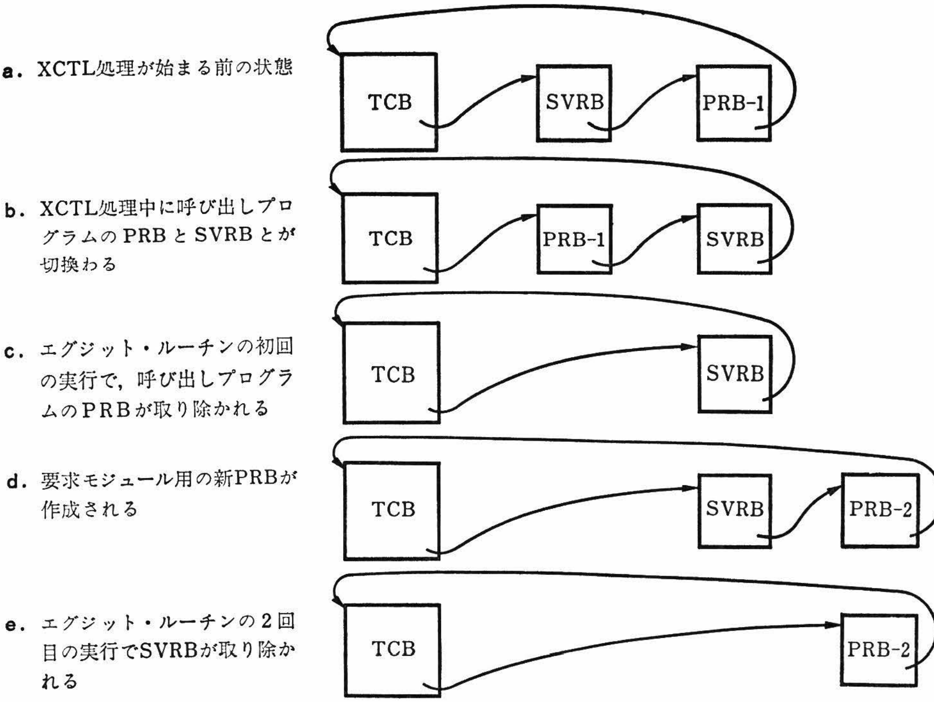
1) LOAD の最終的処理機能

LOAD 要求の最終的処理は，要求モジュールが主記憶装置に入り使用可能になったあとで行なわれる．

内容監視プログラムは，所定のロード・リストを探索して，該当モジュールに対する LLE があるかどうか調べる．そのモジュールの LLE が存在しないと，新規に LLE を作成し呼び出しタスクの TCB でポイントされているロード・リストに入れる．LLE を作成したあと，あるいはすでに LLE が存在する場合には，LLE の中の責任回数フィールド（LLCOUNT フィールド）に 1 を加え，新たな LOAD 要求が附加されたことを示す．このあとで，エグジット・ルーチンに制御権を渡す．

2) XCTL の特殊な処理機能

呼び出しタスクで XCTL マクロ命令を出すと，XCTL の特殊な処理機能が遂行される．マクロ命令がユーザー・プログラムないしはユーザー・エグジット・ルーチンで出された場合だと，制御権が内容監視プログラムのうちの共通サブルーチンに渡る前に特殊処理が行なわれる．一方，XCTL マクロ命令が SVC ルーチンで出されると，特殊処理は**一時域処理ルーチン**（transient area handler）で遂行される．一時域処理ルーチンは所要の SVC ルーチンをリンクして特殊処理を実行する方法をとり，内容監視プログラムの共通サブルーチンを用いないのである．



凡例：
SVRB：内容監視プログラム用SVRB
PRB-1：XCTL命令を出したユーザー・プログラム用PRB
PRB-2：XCTL命令で要求したモジュール用の新PRB

図 8-69　RB 待行列の処理——ユーザー・プログラムで XCTL 要求を出した場合

以下でユーザー・プログラムまたはユーザー・エグジット・ルーチンで XCTL マクロ命令が出された場合と，SVC ルーチンで出された場合とに大別してそれぞれの処理内容を説明してゆく．

①　ユーザー・プログラムまたはユーザー・エグジット・ルーチンで XCTL マクロ命令を出した場合

この場合には，XCTL マクロ命令を出したプログラムの RB を取り除かなければならない．というのは，XCTL マクロ命令でモジュールを呼び出した場合には，そのモジュールの処理が終了したあとで，XCTL マクロ命令を出した元のプログラムに制御権を戻さないようにする必要があるからだ．ともあれ，

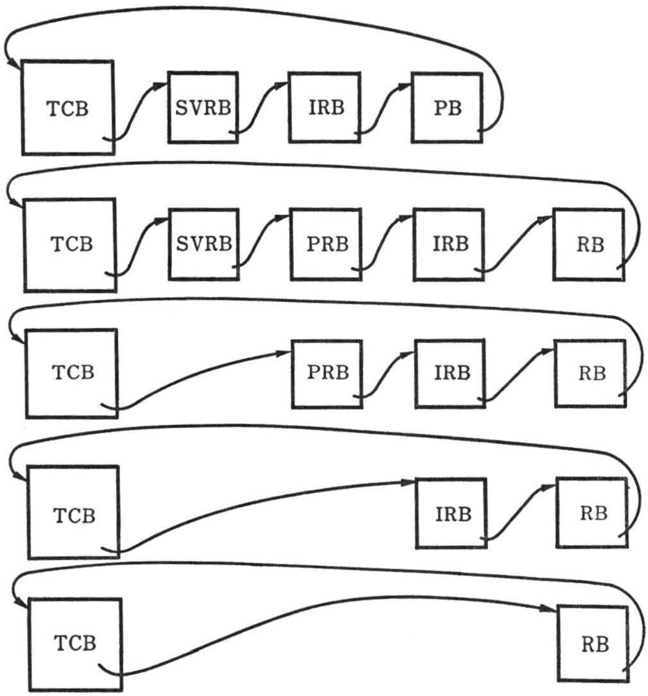
a．XCTL処理が始まる前の状態

b．要求モジュール用の新PRBが作成される

c．内容監視プログラムの実行が完了すると，エグジット・ルーチンがSVRBを取り除く

d．要求モジュールの実行が終ると，エグジット・ルーチンが要求モジュールのPRBを取り除く

e．ディスパッチャーがユーザー・エグジット・ルーチンの再試行をやったあと，エグジット・ルーチンが呼び出しプログラムのIRBを取り除く



凡例：

SVRB：内容監視プログラム用SVRB
RB　：非同期的事象が発生したときに実行されていたルーチンのRB
PRB　：XCTL命令で要求したモジュール用の新PRB
IRB　：ユーザー・エグジット・ルーチン用IRB

図 8-70 RB 待行列の処理――ユーザー・エグジット・ルーチンで XCTL 要求を出した場合

RB を取り除くのにはエグジット・ルーチンが用いられる．

ⓐ **ユーザー・プログラムで XCTL 要求を出した場合**（図 8-69 を参照のこと）

ユーザー・プログラムで XCTL マクロ命令が出された場合，換言するなら PRB のもとでプログラムが実行されている場合には，要求モジュールが実行される直前にまずその PRB が取り除かれる．

該当 PRB を取り除くために，RB 待行列上で呼び出しプログラムの PRB

と内容監視プログラムのSVRBとが置換され，PRBが待行列の最初にくるようにする（図8-69のb. を参照）．また，SVRBの旧PSWフィールド（RBOPSWフィールド）には内容監視プログラムの探索ルーチンのアドレスをセットし，ここからXCTLの処理が再開できるようにしておく．その後で，エグジット・ルーチンにとび，このルーチンで呼び出しプログラムのPRBは取り除かれる（図8-69のc. を参照）．再びこのタスクが制御権を得ると，内容監視プログラムの探索ルーチンから処理が再開され，要求されたモジュールの新PRBが作成されRB待行列に置かれる（図8-69のd. を参照）．さらに，内容監視プログラムから再びエグジット・ルーチンに制御権が移り，内容監視プログラム用のSVRBを取り除くことによって，要求モジュールの実行が可能となる（図8-69のe. を参照）．

ⓑ　ユーザー・エグジット・ルーチンでXCTL要求を出した場合（図8-70を参照のこと）

ユーザー・エグジット・ルーチンでXCTLマクロ命令が出された場合，つまりIRBのもとでプログラムが実行されている場合には，XCTLマクロ命令を出したプログラムのPRBは，要求モジュールの実行が終了したときにはじめて取り除かれる．というのは，IRBには非同期的事象によって中断されたプログラムのレジスターの内容を保存しているので，IRBを直ちに取り除くことができないためである．

XCTLマクロ命令を出したユーザー・ルーチンのIRBの中の旧PSW（RBOPSW）にエグジット・ルーチンのアドレス（実際には，CVTの中のSVC 3命令のアドレスを適用）を入れてから，内容監視プログラムの実行を続行する（図8-70のb. とc. を参照）．これにより，要求されたモジュールの実行が終了するとSVC 3命令が実行されてエグジット・ルーチンに制御権が移され，そこではじめてIRBが取り除かれる（図8-70のd. とe. を参照）．

②　SVCルーチンでXCTLマクロ命令を出した場合

SVCルーチンでXCTL要求を出した場合，つまりSVRBのもとでプログラムを実行している場合には，一時域処理ルーチンの中のXCTLルーチンが

特殊処理を遂行するので，内容監視プログラムの共通サブルーチンは用いられない．一時域処理ルーチンの中の XCTL ルーチンはつぎのことを行なう：

- XCTL マクロ命令を出したSVCルーチンに対応する SVRB を取り除くことによって一時域待行列を更新する．
- 要求されたルーチンが連係パック域にあるかどうか調べ，あれば制御権を渡す．
- 要求されたルーチンが**一時域ブロック**（Transtent Area Block；略して **TAB**）にあるかどうか調べる．
- もし，TAB に該当ルーチンがあれば，それに制御権を渡す準備をする．
- 逆に，TAB に該当ルーチンがなければ，使用可能な TAB を捜す．
- TAB が使用可能でないと，要求を一時延期する．
- TAB が使用可能であると要求ルーチンを TAB へフェッチする準備をする．
- 要求ルーチンをロードする．

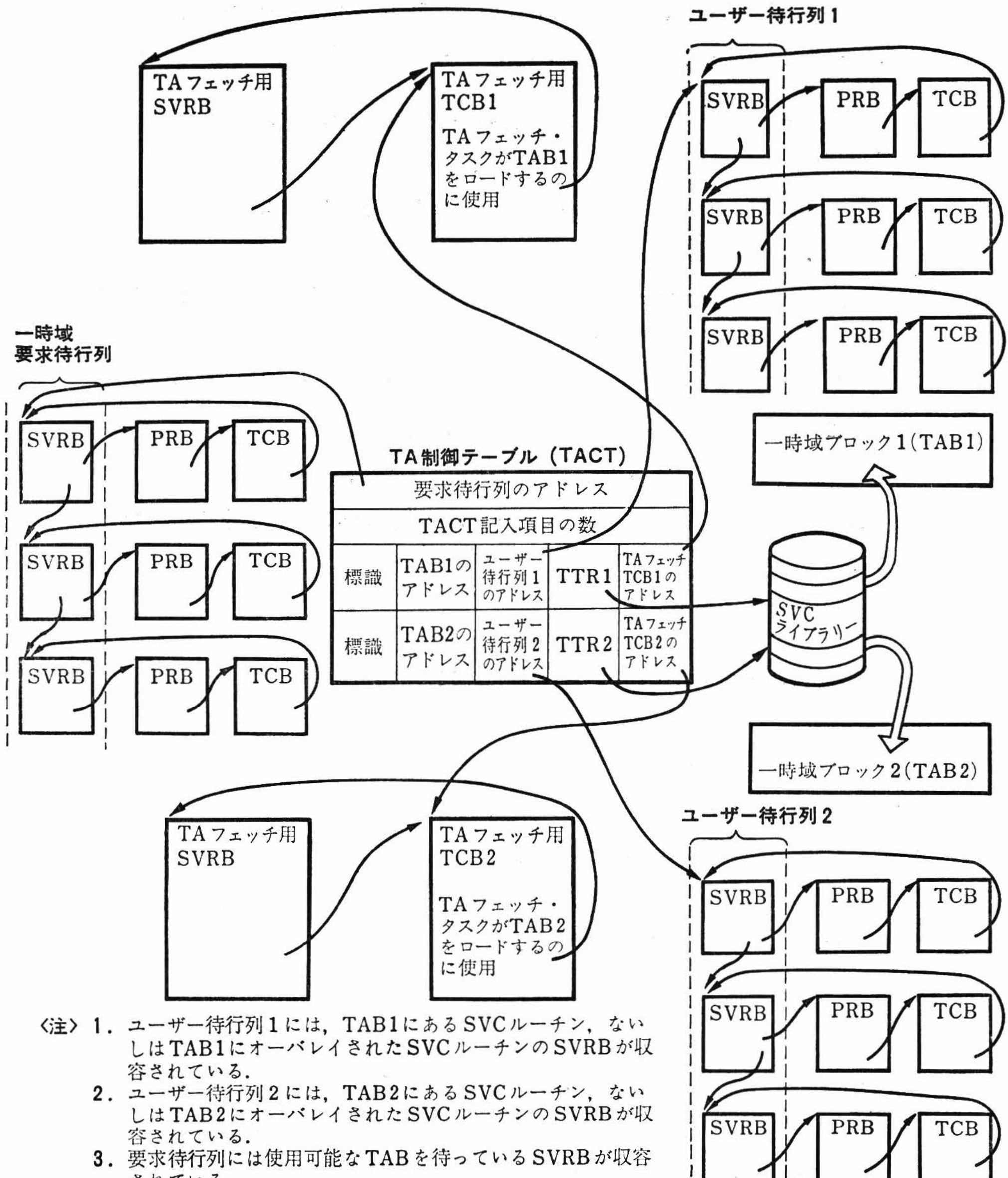
以下 XCTL でルーチンのそれぞれの機能に関する補足説明をしておこう．

ⓐ ユーザ待行列の更新

一時域 XCTL ルーチンは，まず XCTL マクロ命令を出したSVCルーチンが常駐 SVC ルーチンか，非常駐 SVC ルーチンかどうか調べる．その結果，非常駐 SVC ルーチンであれば，その SVRB を TAB のユーザー待行列（**図 8-71** を参照）から取り除き，ユーザー待行列を更新する．常駐 SVC ルーチンだとこの処理は行なわない．

ⓑ 連係パック域のルーチンのテストと制御権の受渡し

ⓐの処理を行なったあとで，XCTL ルーチンは要求されたルーチンが連係パック域にあるかどうか調べる．連係パック域にあると SVRB の属性フィールド（RBSTAB）の "resident" 標識をセットする．さらに，レジスターの内容をセットし，呼び出しルーチンの SVRB の中の旧 PSW フィールド（RBOPSW フィールド）に要求されたルーチンのアドレスを入れたあと，SVC 3 命令を使ってエグジット・ルーチンに制御権を移す．エグジット・ル



〈注〉 1．ユーザー待行列1には，TAB1にあるSVCルーチン，ないしはTAB1にオーバレイされたSVCルーチンのSVRBが収容されている．
2．ユーザー待行列2には，TAB2にあるSVCルーチン，ないしはTAB2にオーバレイされたSVCルーチンのSVRBが収容されている．
3．要求待行列には使用可能なTABを待っているSVRBが収容されている．

図 8-71　一時域待行列

ーチンでは RB 待行列上の最初にある RB (つまり, XCTL マクロ命令で作られた SVRB) を取り除き, その次にあるRB (つまり, XCTL マクロ命令を出したルーチンの SVRB) を最初の RB とする. この結果, XCTL マクロ命令を出した SVRB が要求されたルーチンの実行を制御するようになる. つまり, この SVRB が要求されたルーチンの SVRB の代りをするのであり, みかけ上は XCTL マクロ命令を出した SVC ルーチン (呼び出しルーチン) の SVRB が RB 待行列から取り除かれたと同じになる. つぎに, ディスパッチャーを通して要求されたルーチンに制御権が移される.

ⓒ 要求ルーチンが一時域ブロックにあるかどうかの判定

ⓑで LPACQ を捜しても, 要求されたルーチンが見つからない時には, そのルーチンは非常駐であるとみなされる. したがって, 一時域処理ルーチンの中の XCTL ルーチンは要求ルーチンがどれかの TAB にあるかどうか調べる. このために一時域フェッチ・タスク用の SVRB を調べ, 要求されたルーチン名がみつかれば, そのルーチンが TAB に存在することになる. この場合には, TAB にある SVC ルーチンへの連係を準備する.

ⓓ 一時域ブロックに要求ルーチンがある場合の処理

要求ルーチンが TAB にあると, XCTL ルーチンはその TAB にあるルーチンの実行中にそれがオーバレイされる場合にそなえて, その TAB を回復するのに必要な情報を呼び出しルーチンの SVRB に貯える. つまり, 該当ルーチンが収容されている補助記憶装置上のルーチンのアドレス (TTR), 該当ルーチンのサイズ, 該当ルーチン名の右4文字, その TAB のための**一時域制御ブロック** (transient area control block) の始点から SVRB で表示されているルーチンの入口点までの変位を SVRB のそれぞれのフィールドに貯えておく. そのあとで, TAB のユーザー・カウントに1を加え, 呼び出しルーチンの SVRB をユーザー待行列におく.

非常駐ルーチンへ制御権を渡すための準備は, 連係パック域にある常駐ルーチンの場合と同じである.

ⓔ 一時域ブロックに要求ルーチンがない場合の処理

ⓒで一時域タスク用の SVRB を調べても, 要求された SVC ルーチンの

名前がない時にはそのルーチンが TAB には存在していないことを意味する．そこで，XCTL ルーチンは，要求されたルーチンをどの TAB にローディングできるかの判定に入る．

このために，一時域制御テーブル（図 8-71 を参照）とユーザー待行列を調べ使用可能な TAB があるかどうかを判定する．つぎの場合には TAB が使用可能である：

- TAB が使われていない．
- TAB に作動可能状態の SVRB がない．
- TAB が要求 SVC ルーチンによってオーバレイ可能である（TAB は，現在使用中のタスクの指名順位が要求タスクのそれよりも低いときにオーバレイ可能とみなされる）．

一時域制御テーブルとユーザー待行列とを調べた結果，いずれの TAB も使用不可能であると，その要求は待される．使用可能であれば，要求されたルーチンをその TAB へロードし，ローディングが完了したのちディスパッチャーを通じてローディングされたルーチンに制御権が移される．

ⓕ　要求の延期

どの TAB も使用不可能であると，一時域 XCTL ルーチンは呼び出しルーチンの SVRB を一時域要求待行列（図 8-71 を参照）に入れて置くことにより，その要求を待たせておく．

ⓖ　一時域制御ブロックのオーバレイ準備

ⓔで使用可能な TAB が見つかったら，一時域 XCTL ルーチンは要求 SVC ルーチンをその TAB へフェッチする用意をする．具体的には，その TAB のユーザー待行列にある SVRB，つまりその TAB にあるルーチンを目下使用している SVRB を待ち状態にする．さらに，要求された SVC ルーチンが TAB へローディングされるまで，その実行を待たせておくために呼び出しルーチンの SVRB も待ち状態にする．次に，その SVRB の旧 PSW フィールド（RBOPSW フィールド）に該当TABのアドレスをセットする．また，一時域制御テーブルの標識フィールドにその TAB は現在ローディン

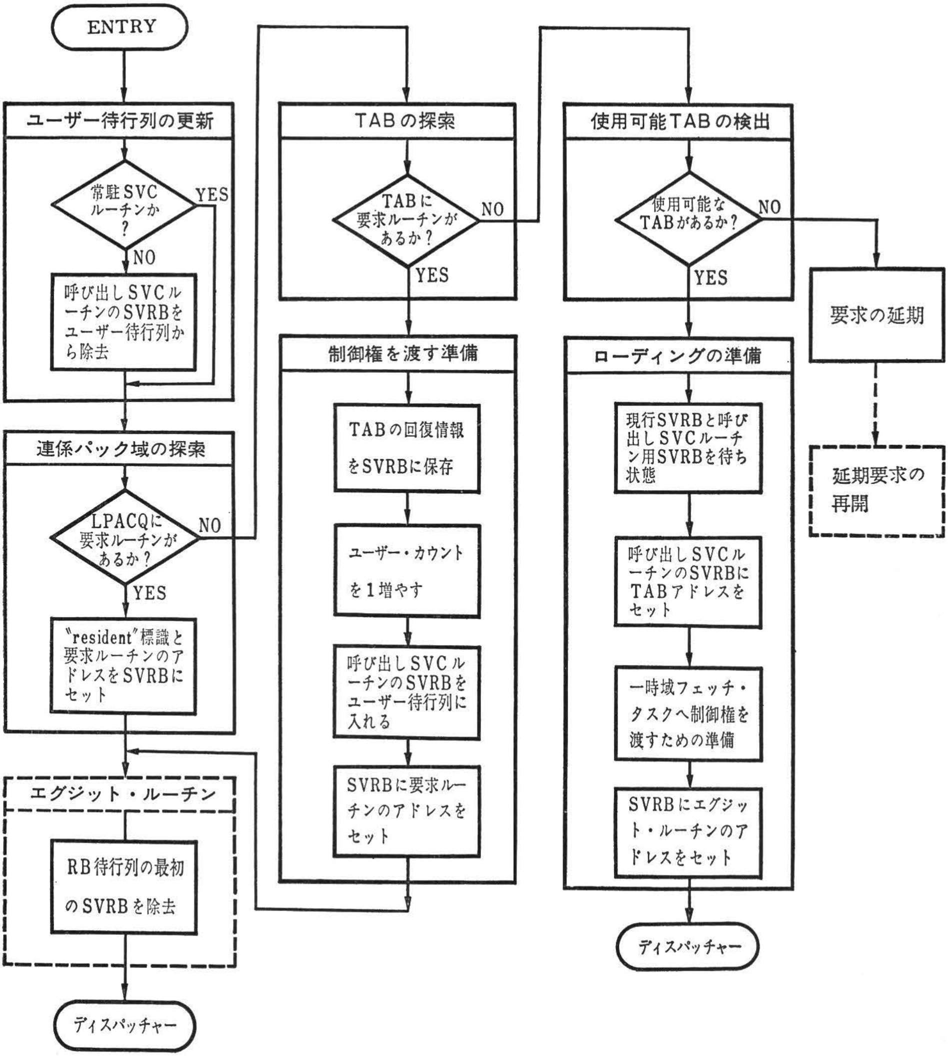


図 8-72　一時域 XCTL ルーチンでの制御権の流れ

グ中であるという標識を立てる.

このあとで，一時域フェッチ・タスクへ制御権を渡す準備をする．さらに，この XCTL 要求で作られた SVRB を取り除くために，その SVRB の旧 PSW フィールド（RBOLD フィールド）にエグジット・ルーチンのアド

表 8-19　IDENTIFY マクロ命令の一般形式

〔記号〕	IDENTIFY	{EP=記号 / EPLOC=名前アドレス} ,ENTRY=入口点アドレス

EP=；入口点の名前
EPLOC=；入口点が収容されている記憶域のアドレス
ENTRY=：追加すべき入口点の記憶域アドレス

レスをセットして，ディスパッチャーへブランチする．

以上の一時域処理ルーチンの中の XCTL ルーチンの 機能を制御の流れにもとづいてまとめたのが**図 8-72** である．

3) IDENTIFY 機能

IDENTIFY マクロ命令の目的は，連係編集プログラムで作った**入口点名**(entry point name)，つまりモジュール名と別名以外の入口点名を作ることである．これによって，動的に入口点名を追加することができるようになる．また，1つのモジュールで17個以上の入口点をもつことができるようになり，モジュールの任意の点に制御権を渡すことが可能となる．なお，IDENTIFY マクロ命令は**表 8-19** に示した通りである．

IDENTIFY マクロ命令が実行されると，制御権は identify SVC ルーチンに渡る． identify SVC ルーチンは，連係編集プログラムが作りあげた以外のモジュールの入口点名を監視プログラムに知らせる．実際には，その入口点名を表示する CDE を別に作成し，それによって監視プログラムに新しい追加の入口点名を知らせるのである．ともあれ，identify SVC ルーチンはつぎの順序で機能する．

identify ルーチンは，まず IDENTIFY マクロ命令を出したプログラムが妥当であるかどうか調べる． IDENTIFY マクロ命令はユーザー・プログラムでしか使用することができないので，マクロ命令を出したプログラムの RB が PRB でないとその要求は妥当でないことになる．この場合には，X “10” の戻

りコードを返す．

要求を出したプログラムの RB が PRB であれば，次に IDENTIFY マクロ命令で指定された入口点名がすでに存在しているかどうかを調べる．このために，identify ルーチンは，指定された入口点と JPACQ および LPACQ を構成している CDE の中の入口点名とを比較する．CDE の CDNAME フィールドで表示されている名前は，モジュールの本名（つまりモジュール名）である場合，モジュールの別名である場合，さらには先行する IDENTIFY マクロ命令で指定された入口点名である場合が考えられる．もしも，これらのいずれかと同じ入口点名だと誤りとみなされるので，その旨要求プログラムに知らせる．

指定された入口点名が幸いにして内容登録簿内になかったなら，つぎには，IDENTIFY マクロ命令で指定されたアドレスが妥当かどうか調べる．このために，IDENTIFY ルーチンは，そのアドレスが呼び出しモジュール内にあるかどうか調べ，もしなければ，そのモジュールのタスクでロードした別のモジュール内にあるかどうか調べる．このテストのためには，LLE, CDE, XL が使われる．テストの結果，アドレスがこれらのモジュール内でないときは戻りコード X "OC" を要求プログラムに返す．

入口点名が内容登録簿になく，かつ入口点アドレスが妥当であると，identify ルーチンはその入口点を表示した副 CDE を作成し，そのモジュールの主 CDE とチェイニングする．そのあとで，探索結果を示す戻りコードを準備したうえで，エグジット・ルーチンとディスパッチャを経由して，呼び出しプログラムへ制御を戻す．

4) DELETE 機能

DELETE マクロ命令は，LOAD マクロ命令でローディングされたモジュールがもはや主記憶装置内に存在する必要がないことを監視プログラムに知らせる．このために機能するのが Delete SVC ルーチンであり，システム・プログラムまたはユーザー・プログラムで利用される．

Delete SVC ルーチンは，該当タスクのロード・リストを探し，要求されたモジュールに対応する CDE を見つける．見つからなければ，当然のことなが

らエラー・コードを要求を出したモジュールにもどす．該当 LLE が見つかったら，LLCOUNT フィールドから1を引いたうえで，そのカウントがゼロになったかどうか調べる．ゼロになっているとそのモジュールに対する LOAD 要求の回数に等しい DELETE 要求が出されたことを意味し，もはやそのモジュールに対する LOAD 要求は残されていないので，その LLE を解放する．

つぎに，CDE の CDUSE フィールドから1を引き，ゼロになったかどうか調べる．このカウントは，そのモジュールに対する要求でまだ完了していないものの合計を示している（単に LOAD 要求だけでなく，LINK，XCTL，ATTACH 要求も含めた合計を示している）．このカウントがゼロであると，そのモジュールに対する要求はもう残っていないことになるので，モジュール，それに対応する CDE と XL をすべて解放する．

5）オーバレイ監視プログラムの機能

オーバレイ監視プログラムは，オーバーレイ・モジュールのローディングを指示する．オーバーレイ・モジュールの実行に先立って，連係編集プログラムが2種類のテーブルをお膳立てしてくれる．つまり，**セグメント・テーブル**（segment table；略して **SEGTAB**）と**エントリー・テーブル**（entry table；略して **ENTAB**）を作りあげ，オーバレイ・モジュールの所定の個所へ入れてくれる．後刻そのモジュールを実行するさいに，オーバレイ監視プログラムはその機能を遂行するためにテーブル内の情報を利用したり，場合によっては情報を変更したりするのである．

①　連係編集プログラムによる前準備機能

先述したように，連係編集プログラムは，目的モジュールないしはロード・モジュール内にある**再配置登録簿**（Relocation Dictionary；略して **RLD**）とユーザーが指定した制御ステートメントの中の情報を用いてSEGTAB と ENTAB を作りあげる．これらのテーブルはオーバレイ・モジュールの一部を構成し，オーバレイ監視プログラムが該当モジュールを実行するさいに活用する．なお，連係編集プログラムが出力したオーバレイ・モジュールの構成は図8-73に示した通りである．

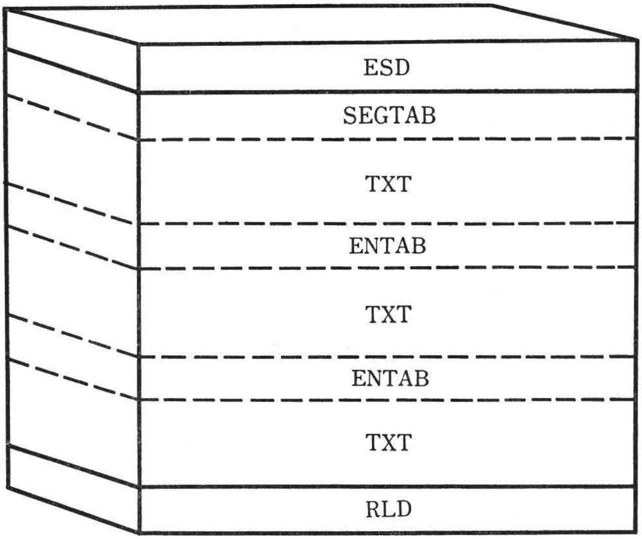


図 8-73　オーバレイ・モジュールの構成

図 8-73 で示したように，オーバレイ・モジュールには唯一の SEGTAB しかない．このテーブルは，モジュール内のセグメントの相互関係を保持するのに使われ，どのセグメントが主記憶装置内に存在し，どのセグメントがローディング中であるかを判定するのに使われる．一方，ENTAB は V-タイプ・アドレス定数をもったセグメントごとに作られる．ENTABの記入項目は，V-タイプ・アドレス定数ごとに構成され，該当セグメントが CALL またはブランチ命令でフェッチされる記号名を表示している．連係編集プログラムは，各記入項目の中に該当定数に割り当てた値を保存する．また，定数の値フィールドに ENTAB 記入項目のアドレスを挿入してくれる．

モジュールの実行中，アドレス定数を使用するブランチ命令が実行されると，そのアドレス定数に関係する ENTAB 記入項目内にある命令に制御権を移す．ENTAB 内の命令は，要求セグメントが主記憶装置にないときにオーバレイ監視プログラムと監視プログラム連係をはかる便宜を提供する．そのセグメントがオーバレイ監視プログラムでフェッチされると，ENTAB 内の命令はそのセグメントへのブランチの便宜をはかるのである．

② オーバレイ監視プログラムの機能

オーバレイ監視プログラムは，つぎの2つの状態のいずれかで制御権を得る：

- オーバレイ・セグメントが別のセグメントに対してSEGLDまたはSEGWT要求を出したとき．
- オーバレイ・セグメントで主記憶装置にない別のセグメントの**外部アドレス**(external address)をCALLする命令，またはブランチする命令を出した時，

いずれの場合にも，オーバレイ監視プログラムは要求セグメントがすでに主記憶装置に存在するかどうか，そのセグメントへのすべてのパスがローディングずみかどうかをSEGTABを調べて判定する．その結果，要求セグメントやパス上の必要セグメントが主記憶装置に存在していないとなると，そのセグメントのローディングを行なう．この場合，実際のローディングはプログラム・フェッチ・ルーチンで遂行する（後述）．

ローディングが完了して，呼び出しセグメントでCALLまたはブランチ命令を出すと，監視プログラムはロードされたセグメントに対応するSEGTABを変更する．修正されたENTABを用いることによって，これ以降ロードされたセグメントの同じ場所へブランチする場合には，オーバレイ監視プログラムの介入なしに行なうことができる．

最後に，オーバレイ監視プログラムは制御権を返すことになるが，どこへ制御権を戻すかは，セグメントを呼び出したマクロ命令のタイプによって異なる．SEGLDマクロ命令の場合だと，ローディングが完了する前に呼び出しセグメントに制御権が戻されるし，SEGWTマクロ命令の場合だと，ローディングが完了したあとで呼び出しセグメントに制御権が戻される．また，CALLまたはブランチ命令でセグメントがローディングされた場合には，ローディングが終了したあと要求セグメントのブランチ・アドレスに制御権が渡されるのである．

オーバレイ監視プログラムの大まかな機能は以上の通りであるが，これら機能について若干補足しておこう．

③　オーバレイ監視プログラムとの連係機能（図8-74）

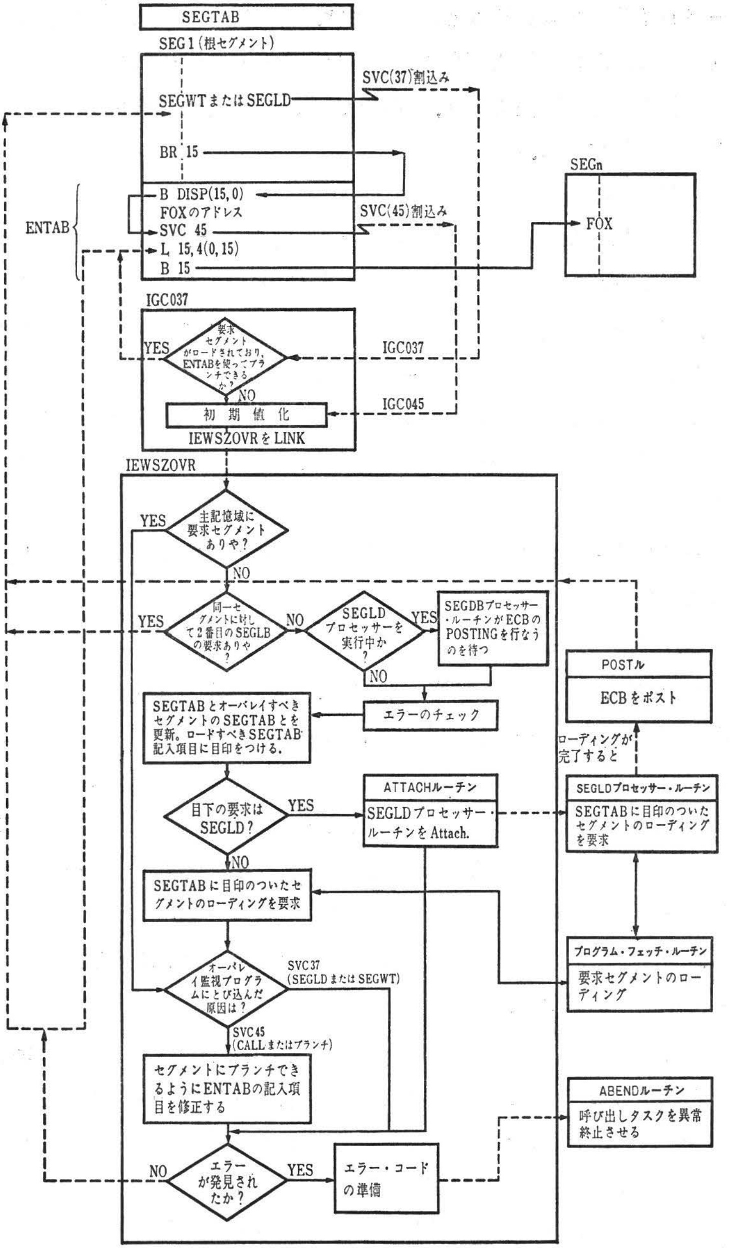


図 8-74 オーバレイ監視プログラムの機能的流れ

オーバレイ監視プログラムとの連係には，**直接連係**（direct linkage）と**間接連係**（indirect linkage）とがある．前者は SEGLD または SEGWT マクロ命令が出された場合の連係であり，後者は CALL またはブランチ命令が実行されたときの連係である．

ⓐ　**直接連係の場合**

SEGLD または SEGWT マクロ命令が出されると，SVC 37 の割込みによって直接オーバレイ監視プログラムの IGC 037 モジュールに制御権が渡される．要求を出したセグメントの ENTAB を通じて必要なセグメントへ直接ブランチするためのエントリーがすぐに用意されていると，制御権は直ちに元のセグメントの ENTAB の所定場所へ戻される．まだ用意されていなければ，IEWSZOVR ルーチンを連係する．

ⓑ　**間接連係の場合**

オーバレイ・セグメントの中で V-タイプ・アドレス定数を指定し，CALL またはブランチ命令が実行されると，ENTAB 内の記入項目へのブランチとなり，さらにそこから ENTAB 記入項目の最後にある SVC 45 の命令へブランチする． SVC 45 の割込みにより，制御権はオーバレイ監視プログラムの IGC 045 モジュールに渡される． IGC 045 モジュールでは IEWSZOVR モジュールに対して LINK マクロ命令を出して制御権を渡す．

④　**ロードすべきセグメントの判定機能**（図 8-74 と**図 8-75** を参照のこと）

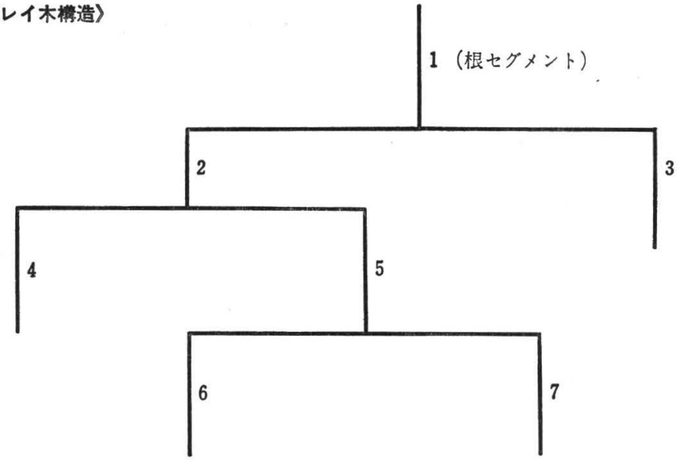
IEWSZOVR ルーチンでは，SIEGTAB の状況インディケーターを調べ，要求されたセグメントへのパスにどのセグメントをロードしなければならないか決める．その結果，ロードしなければならないセグメントに対しては，インディケーターをセットする．

なお，SEGTAB は連係編集プログラムによって作られ，根セグメントの1部に組み入れられており，オーバレイ木構造を表示している．この例を示したのが図 8-75 である．また，SEGTAB 記入項目の状態とそれにもとづいた IEWSZOVR の処理内容をまとめたのが**表 8-20** である．

⑤　**必要セグメントのローディングの制御機能**（図 8-74 を参照のこと）

必要セグメントのローディングの仕方は，SEGWT 要求と SEGLD 要求の場

《オーバレイ木構造》



《木構造に対する SEGTAB 記入項目》

先行セグメント・フィールドの数 ↓　　状況インディケーター ↓

セグメント 1	0	ゼロ	
2	1	ENTAB記入項目のアドレス	
3	1	ENTAB記入項目のアドレス	
4	2	ENTAB記入項目のアドレス	
5	2	ENTAB記入項目のアドレス	
6	5	ENTAB記入項目のアドレス	
7	5	ENTAB記入項目のアドレス	

図 8-75　オーバレイ構造に対応する SEGTAB 記入項目の構成

合で異なる.

ⓐ **SEGWT 要求**

SEGWT マクロ命令による要求に対しては, IEWSZOVR ルーチンは SEGTAB 記入項目の中で目印のついたセグメントをローディングするために直接プログラム・フェッチ・ルーチンを呼び出す. すべてのセグメントがロードされたあとで, 呼び出しセグメントに制御権が戻される.

ⓑ **SEGLD 要求**

表 8-20　セグメント・テーブル（SEGTAB）記入項目の処理

状　　態	IEWSZOVR の処理
1. 要求セグメントが主記憶域に存在する（状況インディケーター"10"）	記入項目が SEGWT または SEGLD 要求に対応するものであると制御権は呼び出しセグメントに戻される．記入項目が CALL またはブランチに対応するものであると，ENTAB 記入項目が変更され，セグメントにブランチするためのエントリーが用意される．
2. 要求セグメントが主記憶域に存在しない（状況インディケーター"11"）	"ローディングがスケジュールされた"（01）旨のインディケーターをセットし，探索を続ける．先行記入項目が要求セグメントのパスにあるセグメントに対応するものであるかどうか判定する．
3. 先行記入項目が要求セグメントのパスに存在するセグメントに対応している	先行記入項目の状況インディケーターをチェックし，そのセグメントが主記憶域に存在しているかどうか判定する．（次のステップは5. または6.）
4. 先行記入項目が要求セグメントのパスに存在しないセグメントに対応している	先行記入項目の状況インディケーターを"主記憶域に存在しない"(11) 旨セットし，そのセグメントのオーバレィを準備する．探索は続行．
5. 先行記入項目がパス上のセグメントに対応しており，そのセグメントが主記憶域にあることを示している．	探索は中止される．要求セグメントのパス内のすべてのセグメントが主記憶域にあるものとみなす（ただし，要求セグメントそのものを除いて）．
6. 先行記入項目がパス上のセグメントに対応しており，そのセグメントが主記憶域にないことを示している．	パス上にあるセグメントの記入項目の状況インディケーターを"ローディングがスケジュールされた"（01）旨セットし，探索を続行する．

SEGLD マクロ命令による要求の場合だと，IEWSZOVR ルーチンはプロセッサー・ルーチンを取り付ける．このルーチンは要求されたセグメントをロードするために，プログラム・フェッチ・ルーチンを呼び出す．制御権は直ちに SEGLD マクロ命令を出したセグメントに返される．

⑥　**ローディングされたセグメントへの直接ブランチの準備機能**（図 8-74 参照のこと）

CALL またはブランチ命令の結果，SVC 45 命仕が出され IEWSZOVR ルー

チンに制御権が渡されると，このルーチンは要求されたセグメントが主記憶装置内にあるかどうか調べ，なければ要求セグメントをローディングする．そのあとで，呼び出しセグメントの中にある ENTAB 記入項目の変位フィールド（DISP フィールド）に2を加算する．

最初に CALL またはブランチ命令が出された時には，DISP フィールドは SVC 45 の命令にブランチするようにセットされていたため，この命令が実行されると IEWSZOVR ルーチンに制御権が渡された．ところが，ここで DISP フィールドに2が加えられることにより，これ以降同じアドレスへ CALL またはブランチ命令が出されるともはや SVC 45 の命令へはブランチしない．その代り，つぎのロード命令（L 命令）へブランチする．これにより，オーバレイ監視プログラムを経由することがなく，直接他のセグメントへブランチすることが可能となる．

以上がオーバレイ監視プログラムの機能である．オーバレイ監視プログラムが遂行する処理のタイプをまとめると**表 8-21** のようになる．

6) プログラム・フェッチ・ルーチンの機能

補助記憶装置から SVC ルーチン，入出力エラー処理ルーチンあるいは非常駐モジュールなどをローディングする時には必ずプログラム・フェッチ・ルーチンが用いられる．プログラム・フェッチ・ルーチンは，必要な主記憶域を確保し，入出力オペレーションを遂行し，必要ならアドレス定数の再配置を行なう．

プログラム・フェッチ・ルーチンが実際に果たす機能は，監視プログラムのどのルーチンがプログラム・フェッチ・ルーチンを要求したかによって異なる．プログラム・フェッチ・ルーチンを呼び出す監視プログラム・ルーチンにはつぎの4種類がある：

- **一時域フェッチ・ルーチン：** 非常駐 SVC ルーチンのフェッチを要求．
- **ステージ3エグジット・エフェクター：** 入出力エラー処理ルーチンを要求．

表 8-21　オーバレイ監視中の処理タイプのまとめ

命　令	状　　態	主　要　な　処　理
SEGLD (SVC 37)	1. 要求セグメントおよび(または)要求セグメント上のパス内のセグメントが主記憶域に存在せず，かつローディング中でもない．	1. 必要セグメントのローディングを開始する．呼び出しセグメントのENTABは，要求セグメントへブランチするための準備をするように修正されることはない．セグメントがローディングされている間に，制御権は呼び出しセグメントに戻される．要求セグメントへ制御権は渡らない．
	2. 要求セグメントが主記憶域にあるか，またはローディング中である．	2. 呼び出しセグメントへ直ちに制御権が戻される．
SEGWT (SVC 37)	3. 1. と同じ状態	3. 必要セグメントをローディングする．呼び出しセグメントのENTABは，要求セグメントへブランチするための準備をするように修正されることはない．制御権が呼び出しセグメントに戻されるのは，要求セグメントおよび，そのパス上の必要セグメントがロードされたときだけである．要求セグメントに制御権は渡らない．
	4. 要求セグメントが SEGLD 要求でローディング中である．	4. SEGWT 要求の処理はローディングが完了されるまで待たされる．何ら新たなローディングは行なわれない．残りの処理は3. と同じである．
	5. 要求セグメントが主記憶域にある．	5. 呼び出しセグメントへ直ちに制御権が戻される．
CALL または ブランチ (SVC 45)	6. セグメントは SEGLD または SEGWT を使って要求され，すでに主記憶域にある．	6. 呼び出しセグメントのENTABは，オーバレイ監視プログラムの助けをかりずに，同じ外部アドレスにブランチするための準備をするように修正される．そのあと制御権は，要求セグメントの指定されたアドレスに受渡される．
	7. セグメントは SEGLD を使って要求されたが，ローディングがまだ完了していない．	7. CALL またはブランチ要求は，ローディングが完了するまで待される．新たなローディングは行なわれない．残りの処理は6. と同じである．
	8. 要求セグメントは主記憶域に存在しないし，ローディング中でもない．	8. 必要セグメントをローディングする．ローディングが完了すると，6. と同じ処理が行なわれる．
	9. 呼び出しセグメントは同じ外部アドレスを指定した CALL またはブランチ命令をすでに出している．	9. オーバレイ監視プログラムは関与しない．呼び出しセグメントの ENTAB は6. と同じ要領で修正されているので，直接要求セグメントにブランチできる．

- **内容監視プログラムの共通サブルーチン：** 主記憶域で使用不可能なモジュールおよびオーバレイ・モジュールのうちの根セグメントを要求.
- **オーバレイ監視プログラム：** 根セグメント以外のオーバレイ・モジュールのセグメントを要求.

以下で，非常駐 SVC ルーチンと入出力エラー処理ルーチンのフェッチ，非常駐モジュール（つまり，通常のモジュールとオーバレイ・モジュール）のフェッチの2つに大別して，プログラム・フェッチ・ルーチンの機能を説明する.

① SVC ルーチンと入出力エラー処理ルーチンのフェッチ

SVC 第2次割込み処理ルーチンまたはステージ3エグジット・エフェクターは，必要とするルーチンが主記憶装置の中の **TAB** (Transient Area Block) にあるかどうか調べる．使用可能なコピーが **TAB** にないと，要求ルーチンを TAB にローディングするために TAB プログラム・フェッチ・ルーチンが呼び出され，そのルーチンに制御権が移される．その結果，非常駐 SVC ルーチンは SVC 用 TAB に，入出力エラー処理ルーチンは入出力監視プログラム用 TAB にローディングされる．この様子を示したのが図 8-76 である.

プログラム・フェッチ・ルーチンを呼び出す時には，呼び出しルーチンで前もってフェッチ作業域にロードすべきルーチンの TTR とサイズとをセットしておく．プログラム・フェッチ・ルーチンでは，この TTR を変換ルーチンを用いて**絶対アドレス** (absolute address) に変換したあとで EXCP マクロ命令を出して，所要ルーチンを補助記憶装置から入力する．EXCP マクロ命令で入出力監視プログラムが呼び出され，実際のフェッチが行なわれるのである.
一時域 SVC ルーチンは，再配置可能なアドレス定数をもたないので，再配置の必要はない.

ともあれ，要求されたルーチンがロードされると，入出力エラーがあったかどうか調べ，戻りコードをレジスターに入れ，呼び出しルーチンへ制御権を戻す.

② 非常駐モジュールのフェッチ

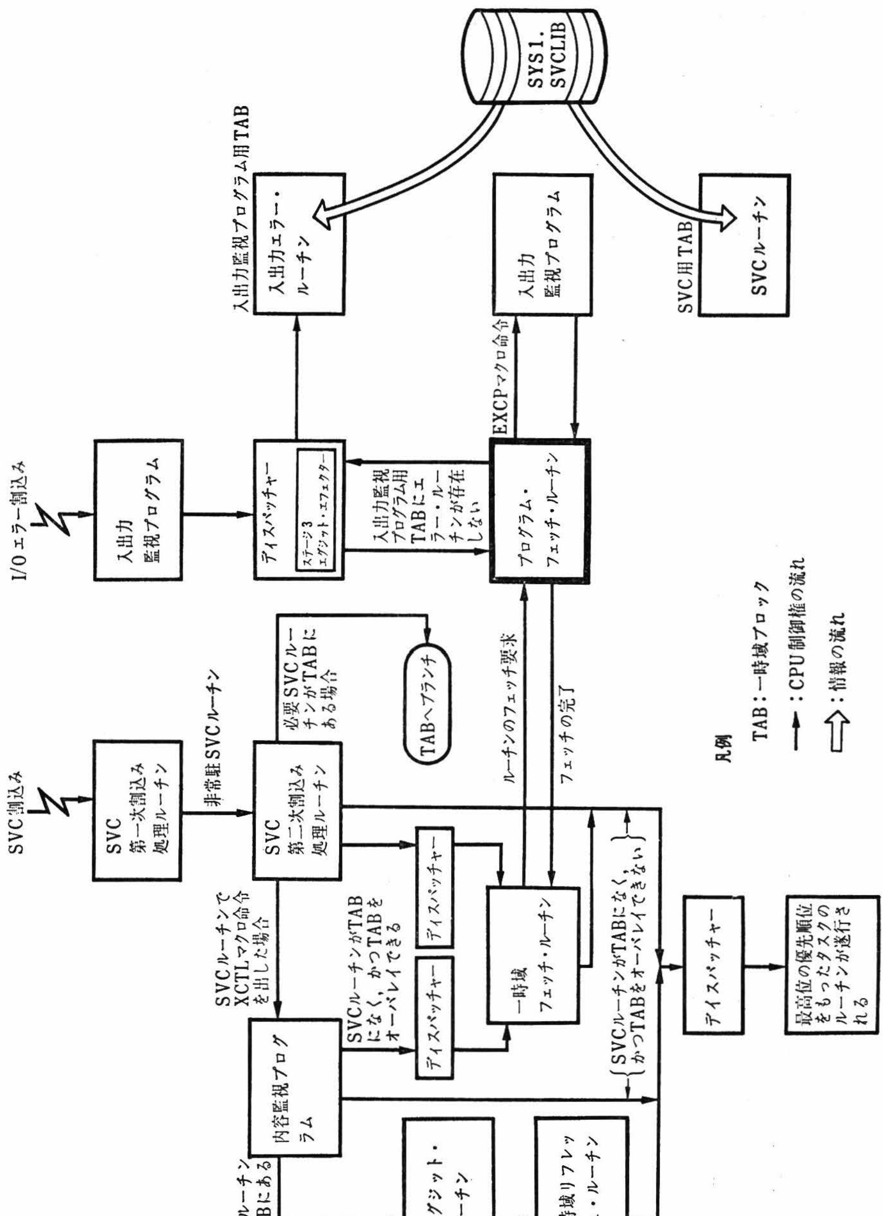


図 8-76　プログラム・フェッチ・ルーチンと他のルーチンの相互関係——SVC ルーチンまたは入出力エラー・ルーチンをフェッチする場合

プログラム・フェッチ・ルーチンは，内容監視プログラムの共通ルーチンまたはオーバレイ監視プログラムのどちらかで呼び出される．必要モジュールが主記憶域に存在しないと，LINK, ATTACH, LOAD, XCTL マクロ命令が出されたあと，内容監視プログラムの共通サブルーチンがプログラム・フェッチ・ルーチンを呼び出すのである．一方，オーバレイ・モジュールの必要セグメントが主記憶域にない場合だと，SEGWT, SEGLD, CALL マクロ命令，またはブランチ命令が出されたあと，オーバレイ監視プログラムがプログラム・フェッチ・ルーチンを呼び出す．この様子を示したのが，**図 8-77** である．

非常駐モジュールないしはオーバレイ・セグメントのローディングにさいしてプログラム・フェッチ・ルーチンが遂行する主要な機能はつぎの４つに集約できる：

- **初期値化**（initialization）：フェッチ作業域を初期値化し，XL を作りあげる．また，モジュールがオーバレイ形式だと，そのモジュールのノート・リストをフェッチする．もしも，モジュールがスキャター・ロードされるのであれば，そのためのスキャター／翻訳テーブルもフェッチする．
- **ローディング**（loading）：補助記憶装置の所定の場所から主記憶装置にテキスト・レコード（TXT の部分）と RLD レコードとを読み込む．テキスト・レコードはローディング・プログラムを構成し，RLD レコードは再配置に用いられる．
- **再配置**（relocation）：ローディングされたプログラムの中のアドレス定数値（これは RLD で示されているのだが）を相対アドレスから絶対アドレスに変換する．
- **終了**（termination）：入出力オペレーションの完了を検査し，再配置されたモジュールの入口点アドレスを計算する．さらに，モジュールがオーバレイ形式の場合には SEGTAB を初期値化する．そのあとで，戻りコードを準備し内容監視プログラムまたはオーバレイ監視プログラムに制御権を戻す．

以上のプログラム・フェッチ・ルーチンの機能を関連する監視プログラム・

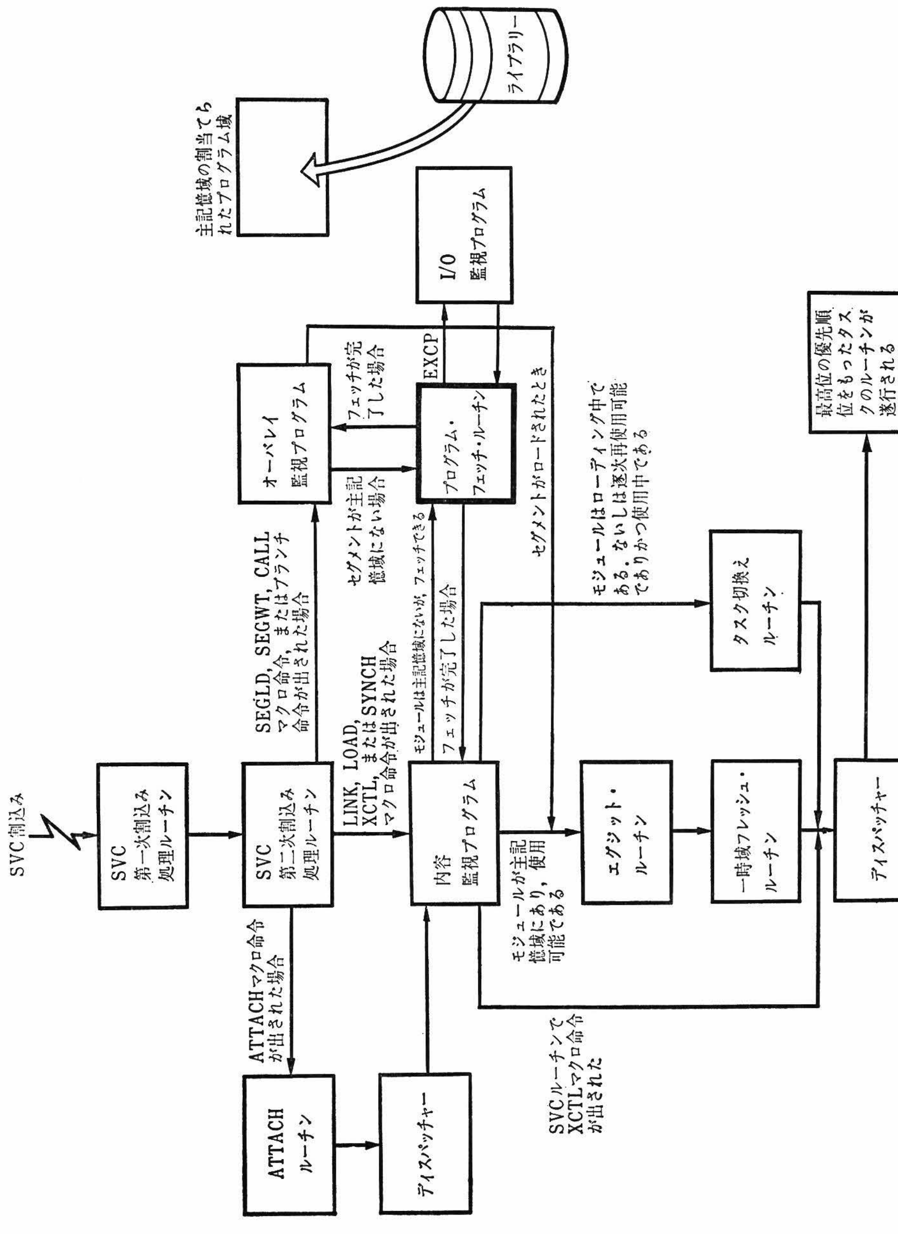


図 8-77　プログラム・フェッチ・ルーチンと他のルーチンの相互関係——モジュールまたはオーバレイ・セグメントをフェッチする場合

ルーチンとの相互関係にもとづいてまとめたのが**図 8-78** である．

六．主記憶装置監視プログラムの機能構造

主記憶域は重要な資源の１つであり，他の諸資源と同様に種々のタスクで共用される．したがって，そのスペースの割当ては一定の基準にもとづいて制御しなければならず，必要のつど要求し，不要になったら解放しなければならない．主記憶域の割当てに関する制御は，主記憶装置監視ルーチンとロールアウト／ロールインのモジュールとで遂行される．しかも，その制御は主記憶域に関する数種類の待行列を用いて行なわれる．

ジョブ管理ルーチンがジョブ・ステップをタスクとして指名すると，そのタスクを遂行するさいに使用する主記憶装置の領域が要求される．領域の大きさはジョブ制御ステートメントで指示した大きさである．領域には該当ステップ用の**ジョブ・パック域**（job pack area）とすべての追加作業域が収容される．

主記憶装置監視ルーチンは２種類のマクロ命令を通じてサービスを提供する．その１つは，GETMAIN マクロ命令であり，スペースの割当てに使われる．もう１つは FREEMAIN マクロ命令であり，すでに割当てずみのスペースを解放するのに用いられる．

したがって，主記憶装置に関するあらゆる要求が，主記憶装置監視プログラムの中の GETMAIN SVC ルーチンで処理される．つまり，領域に対する要求，領域内のスペース要求，ないしはシステム待合せ域内のスペース要求などスペースに対する全要求にこたえるのが GETMAIN ルーチンといえる．一方，主記憶域の解放要求は，主記憶域監視プログラムの構成要素の１つである FREEMAIN SVC ルーチンで処理される．このルーチンは，スペース管理用の制御ブロックを更新し，解放されたスペースの状態を反映することによって，GETMAIN ルーチンがそのスペースを別の要求に再割当てできるようにする．

システム内に１つもジョブ・ステップが存在しないと，動的記憶域全体が１

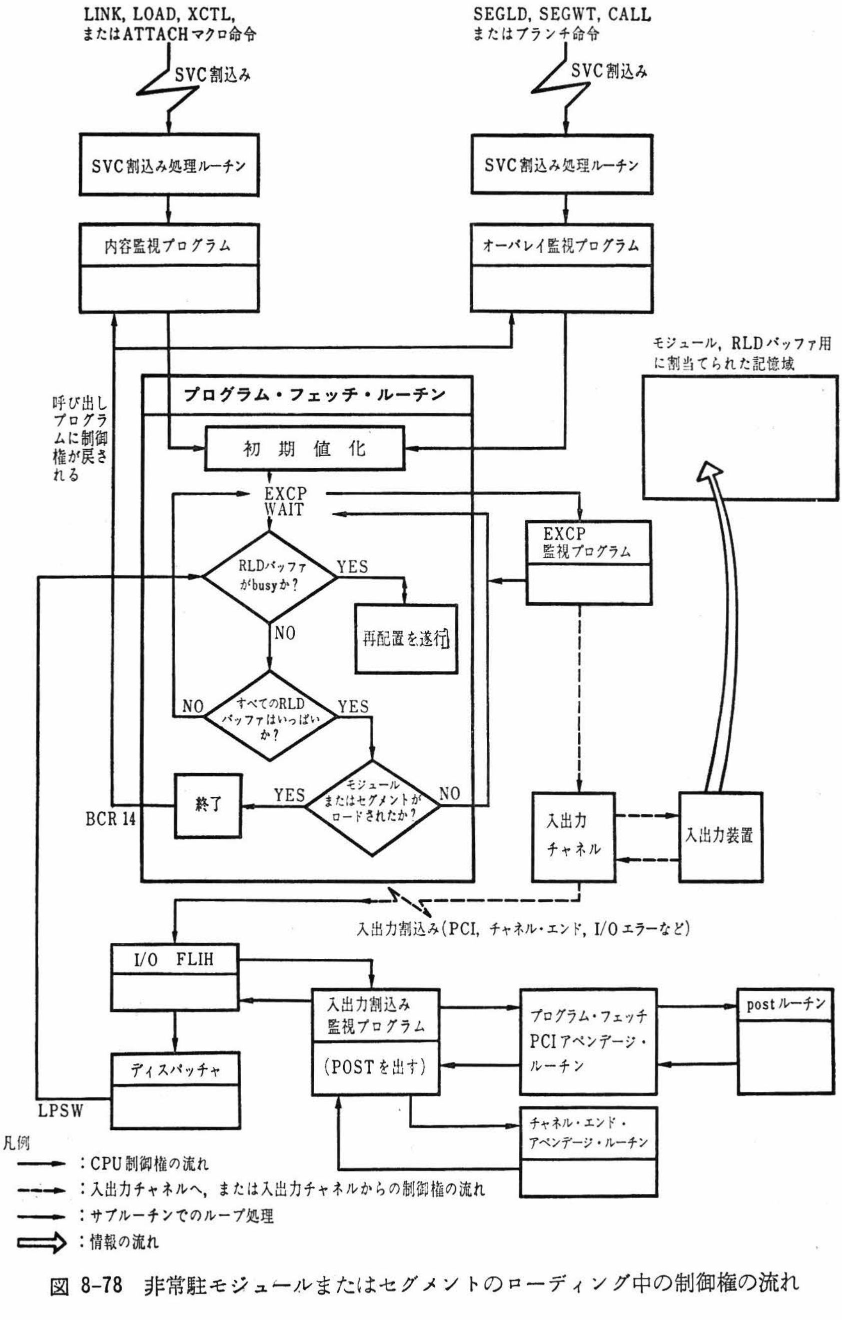


図 8-78　非常駐モジュールまたはセグメントのローディング中の制御権の流れ

個の領域として取り扱われる．これは，領域内の最下位アドレスにもうけられた**フリー・ブロック待行列要素**（Free Block Queue Element；略して **FBQE**）と，システム待合せ域にある**区分待行列要素**（Partition Queue Element；略して **PQE**）とで監視プログラムに知らされる．PQE には FBQE のアドレスが収容されている．換言すると**空きの区域**（free area）の始まりのアドレスが収容されているといえる．一方，FBQE には空きの区域の範囲が示されている．したがって，ジョブ・ステップ用のスペースが要求されると，GETMAIN ルーチンは空きの区域から要求域を確保し，確保した領域のために新規のFBQE と PQE とを作りあげる．また，新規に構築した PQE のアドレスはそのジョブ・ステップ用タスクの TCB 内の所定欄に入れられる．

ジョブ・ステップが開始されたあと，タスクを遂行するプログラムの中でGETMAINマクロ命令を出して記憶域を要求することができる．この場合には，該当ジョブ・ステップにすでに割当てられている領域，またはシステム待合せ域内の記憶域から要求スペースを確保する．監視プログラムは，領域内のスペースの割当てをうまく管理してゆくために待行列要素を相互に連鎖するようにしている．これによって，領域内のサブプールをメインティナンスしているのである．本章の第二節でふれたようにサブプールとは，サブプール番号と呼ばれる目印しをつけて要求した記憶域のことである．サブプール内の記憶域は連続している必要はない．このようなサブプールという考え方の最大の利点は，タスク間で記憶域が共用できることであり，サブプール番号で表示された記憶域の全てを１つの FREEMAIN マクロ命令を使って解放できることにある．

なお，ロールアウト機能が遂行される場合には，ロールアウトによって得られた一時的な領域をジョブ・ステップで要求されたスペースに割当てることが可能となる．

主記憶装置監視プログラムの機能構造を詳述するに先立って，主記憶装置がどのような手法で管理されているのかを概観することから始めよう．

1. 主記憶装置の管理の仕方

すでに再三指摘してきたように，動的記憶域は**領域**(region) に分割される．

領域の大きさは，実行するジョブ・ステップの諸要件に応じたものである．ジョブ・ステップは，第7章でふれたようにジョブ・クラスと優先順位にもとづいて選択され実行される．そのさい，それぞれのジョブ・ステップに対して，JOB ないしは EXEC ステートメントで指示された大きさの領域が割り振られる．万一，最も優先順位の高いジョブ・ステップで要求する領域が，利用可能な記憶域の大きさを越えるとなると，それより優先順位が低くかつ空きの記憶域で十分な領域しか要求していないジョブ・ステップが開始される．いったんジョブ・ステップが開始されると，**ロールアウト／ロールイン** (rollout/rollin) **機能**がシステムに組み込まれているときだけしかその領域を拡大することは望めない．

ともあれ，ジョブ・ステップが開始されたあとそれに割り当てられた領域を利用するためには，記憶域を明示的に要求するか暗示的に要求するかのいずれかの方法をとる．たとえば，LINK マクロ命令を使用すると記憶域の暗示的要求を意味する．監視プログラムは，ジョブ・パック域に要求ロード・モジュールをロードするに先立って，それに必要なスペースを自動的に割当ててくれる．この種の性格のものは暗示的要求である．一方，GETMAIN マクロ命令を用いて一定バイト数の記憶域を要求するとなると，これは明示的要求である．このように通常のプログラムの中で要求する仕方以外に，監視プログラムやデータ管理プログラムが要求する場合もある．ユーザー・タスクを管理してゆくために必要となる各種制御ブロックを収容するための記憶域を確保するなどがこれに当る．

まずは，明示的要求の仕方について説明しよう．

1)　明示的要求 (explicit request)

先述したように，GETMAIN マクロ命令を使うと主記憶域の明示的要求ができる．この場合，該当ジョブ・ステップに対してあらかじめ確保されている領域のうちから，要求された記憶域が活動状態のタスクに割り当てられるのである．重要なことは，記憶域の要求を行なわないでジョブ・ステップ用に確保されている記憶域（つまり領域）の利用はできないことだ．記憶域の要求を

しないまま，それを使用しようとすると必然的にタスクは異常終止してしまう．

一方，要求した記憶域の使用が終わったあとで，その記憶域を解放するのにはFREEMAINマクロ命令を用いる．このマクロ命令を与えることによって，該当記憶域がジョブ・ステップの制御から解放されてしまうわけではない．ジョブ・ステップ内のどれかのタスクから別の記憶域要求が出されたときに，それを満たすのに利用できる状態にされるだけである．なお，あるタスクに割り振られている記憶域は，そのタスクが終了すると自動的に解放され，ジョブ・ステップ内の別のタスクで利用することができるようになる．

まず，GETMAIN マクロ命令による明示的要求の仕方をみておこう．**表8-22**で示したのがこのマクロ命令の一般的形式である．それぞれのパラメータの指定内容と意味とは**表8-23**に示した通りである．

GETMAIN マクロ命令で要求した記憶域は8バイトの倍数の大きさで割り当てられ，しかも倍ワード境界から始まる．要求する記憶域は LV ないしは LA パラメータを使ってバイト単位で指示する．指示するバイト数が8の倍数になっていない場合には，自動的に8の倍数になるように切上げてくれる．必要のつど少量のバイト数でくり返し記憶域を要求することができるし，該当タスクで必要とする記憶を満すに足りる大きな記憶域を一度に要求することもできる．一度に大きな記憶域を要求することの利点は2つ考えられる．1つは，連続した記憶域を確保することができることである．別々に記憶域の要求を出した場合には両者が連続した記憶域となる保障はないが，一回の要求で確保す

表 8-22 GETMAIN マクロ命令の一般形式

〔記号〕	GETMAIN	{ EC,LV=バイト数, A=アドレス[,SP=番号][,HIARCHY=数字] EU,LV=バイト数, A=アドレス[,SP=番号][,HIARCHY=数字] LC,LA=アドレス, A=アドレス[,SP=番号][,HIARCHY=数字] LU,LA=アドレス, A=アドレス[,SP=番号][,HIARCHY=数字] R,LV=バイト数, SP=番号[,HIARCHY=数字] R,LV=(0)[,HIARCHY=数字] VC,LA=アドレス, A=アドレス[,SP=番号][,HIARCHY=数字] VU,LA=アドレス, A=アドレス[,SP=番号][,HIARCHY=数字] }

表 8-23 GETMAIN マクロ命令のパラメーター一覧表

パラメータ	指定の内容	意　　味
E	(element)	SP番号で指示したサブプールから，LV オペランドで表示した長さもった1個の記憶域を確保することを要求.
L	(list)	SP 番号で指示したサブプールから1個以上の 記憶域を確保することを要求. 各記憶域の大きさは，LAオペランドで指示されたアドレスから始まるリスト内の値で表示.
R	(register)	指示されたサブプールから，LV オペランドで指定した長さをもった1個の記憶域を割り当てることを要求. 割り当てられた記憶域のアドレスはレジスター1に戻される.
V	(variable)	ＳＰ番号で指示したサブプールから，ＬＡオペランドで表示したアドレスにある2つの値の間の長さをもった1個の記憶域を確保することを要求.
C	(conditional)	条件付き要求であり，使用可能な記憶域より大きな記憶域を要求しても該当タスクを異常終止しないことを指示.
U	(unconditional)	無条件要求であり，使用可能な記憶域より大きい記憶域を要求した場合には要求タスクは異常終止される.
LV=	バイト数	要求する主記憶域の長さ（バイト数）を指定. 8の倍数であること.
LA=	アドレス	ワード限界から始まる連続したフルワード記憶域のアドレス. 各フルワードには要求する記憶域の長さを入れておくこと.
A=	アドレス	ワード限界から始まる連続したフルワード記憶域のアドレス. Ｅを指定した場合は1ワード，Ｌを指定した場合にはＬＡリストの各エントリーに対応して1ワードずつ必要である. この場所に，確保した記憶域のアドレスが入れられる.
SP=	番号	割り当てるべき記憶域のサブプール番号. 0から127であること. 省略するとサブプール0とみなされる.
HIARCHY=	数字	割り当てる記憶装置のハイアラキー番号を指定（0または1）.

るとなると，確実に連続した記憶域となる．もう一つの利点は，監視プログラムのオーバヘッドが少なくてすむことである．くり返し要求したのでは，そのつどオーバヘッド時間がいくらか費やされてしまうからだ．

ところで，GETMAIN マクロ命令を使って記憶域を明示的に要求するやり方には4種類の方法がある．それぞれの方法は，GETMAIN マクロ命令のオペランドで特別な文字（E, L, R, V）を指定することによって示される．いずれも，

プログラムの要件に応じた利点をもっている．ともあれ，各々の方法について以下で簡単に紹介しておこう：

- **要素タイプ** (E)：LV オペランドで指示した大きさの1個の記憶域を要求することを指定する．割当てられた記憶域のアドレスは，Aパラメータで指示した1ワードの記憶域に入れられる．
- **リスト・タイプ** (L)：1個以上の記憶域を要求することを指示する．各々の記憶域の大きさは，LA パラメータで指示したアドレスから始まる連続したフルワードのそれぞれに入れておく．監視プログラムは，確保した記憶域のアドレスをAパラメータで指示したアドレスから始まる連続したフルワードのそれぞれに入れてくれる．LA パラメータで要求したと同じ順序でAパラメータで指示した場所にアドレスが入れられるのである．
- **レジスター・タイプ** (R)：LV オペランドで指示した大きさの1個の記憶域を要求することを指定する．要素タイプの場合とは異なり，割り当てられた記憶域のアドレスはレジスター1に入れられる．また，このタイプの要求ではパラメータをレジスターを用いて監視プログラムに受渡すので，**再入可能コード** (reentrable code) の形式となる．したがって，再入可能プログラムを作成する場合には好都合となる．
- **可変タイプ** (V)：LA オペランドで指示した2つの値の間の大きさをもった1個の記憶域を要求することを指定する．監視プログラムは，指示された最大の記憶域を確保しようとする．最大の記憶域を満すに十分なスペースが利用可能でないと，2つの値の間の大きさをもった最大の記憶域を割り当てる．確保された記憶域のアドレスと大きさは，Aパラメータで指示した連続したフルワードのそれぞれに入れられる．

このような記憶域の要求に加えて，さらに条件付き要求と無条件要求とがある．ただし，レジスター・タイプの要求では無条件要求しかできない．無条件要求の場合だと，要求を満すに足りる記憶域が利用可能状態でないと，該当タスクは異常終止してしまう．一方，条件つき要求の場合には，十分な記憶域が

利用可能でなくてもタスクは異常終止しない．その代り，レジスター15に**戻りコード**（return code）４がセットされる．要求が満たされたときには０の戻りコードが返される．なお，条件つきのリスト・タイプ要求が出されたときには，要求記憶域のすべてが確保できないかぎり，記憶域の割り当てはおこなわない．

GETMAIN マクロ命令の使用例を示したのが，**図 8-79** である．この例では，16,000バイトの作業域があると最も効率的にプログラムの実行ができると仮定している．8,000 バイト以上ある場合もかなりの効率が期待できるが，4,000 バイトから 8,000 バイトの作業域だときめて効率は悪くなると想定している．また，4,000 バイト以下だとプログラムは実行できない．このプログラムでは，REENTMOD という入口点名をもった再入可能なロード・モジュールを使用しており，プログラムの中で再度利用するものと仮定している．そのために，このロード・モジュールは LOAD マクロ命令でジョブ・パック域に呼び込まれており，必要なときにすぐ利用できる状態にあることを前提としてい

```
              ⋮
              GETMAIN   EC, LV=16000, A=ANSWADD     条件つき要求
              LTR       15, 15                      戻りコードのテスト
              BZ        PROCEED1                    16000バイト割り当てられたときブランチ
              DELETE    EP=REENTMOD                 そうでないとき，主記憶域を解放
              GETMAIN   VU, LA=SIZES, A=ANSWADD     無条件要求
              L         4, ANSWADD+4                割当てられた長さのロード
              CH        4, MIN                      テスト
              BNL       PROCEED1                    8000バイト以上だとPROCEED1へ
PROCEED2      ↓                                     8000バイト未満の場合の処理

PROCEED1      ↓

MIN           DC        H"8000"                     PROCEED1のための最小サイズ
SIZES         DC        F"4000"                     PROCEED2のための最小サイズ
              DC        F"16000"                    最大のサイズ
ANSWADD       DC        F"0"                        割り当てられた記憶域のアドレス
              DC        F"0"                        割り当てられた記憶域のサイズ
              ⋮
```

図 8-79　GETMAIN マクロ命令の使用例

表 8-24 FREEMAIN マクロ命令の一般形式

〔記号〕	FREEMAIN	E, LV=バイト数, A=アドレス〔, SP=番号〕 L, LA=アドレス, A=アドレス〔, SP=番号〕 R, SP=番号 R, SP=(0) R, LV=(0), A=アドレス R, LV=(0), A=(1) R, LV=バイト数, A=アドレス〔, SP=番号〕 R, LV=バイト数, A=(1)〔, SP=番号〕 V, A=アドレス〔, SP番号〕

る．これらのことを念頭にプログラム例をみて欲しい．

最初の GETMAIN マクロ命令では，16,000バイトの大きさの記憶域を条件つきで要求している．要求した大きさの記憶域が利用可能であるかどうかを判定するために，レジスター15の戻りコードをテストしている．戻りコードが0だと16,000バイトの記憶域が割り当てられたことを意味するので，処理ルーチン（PROCEED 1）にブランチしている．十分な記憶域が利用可能でない場合には，DELETE マクロ命令を出して REENTMOD ロード・モジュールが占有しているスペースを解放することによって，割当て可能な記憶域を増やそうとしている．2番目の GETMAIN マクロ命令では，4,000バイトから16,000バイトの間の記憶域の無条件要求をおこなっている．この場合，最小のスペース，つまり4,000バイトのスペースが利用可能でないとタスクが異常終止する．しかし，少なくとも4,000バイトあれば，タスクはそのまま続行できる．ともあれ，実際に割り当てられた記憶域の大きさをテストし，8,000バイト未満の大きさだと PROCEED 2 へ進み，それ以上だと PROCEED 1 にブランチするようにしている．

このようにして GETMAIN マクロ命令で明示的に確保した記憶域が不要になった場合には，FREEMAIN マクロ命令を使って解放する．FREEMAIN マクロ命令も，GETMAIN マクロ命令と同様に，4種類の解放要求の仕方がある．指示の仕方は GETMAIN の場合と変らない．**表 8-24** で FREEMAIN マクロ命令の一般形式を，**表 8-25** でパラメータの指定内容と意味を示している

表 8-25　FREEMAIN マクロ命令のパラメーター一覧表

パラメータ	指定の内容	意　　味
E	(element)	割当てずみの1個の記憶域をSPオペランドで表示されているサブループから解放することを指示.
L	(list)	割当てずみの1個以上の記憶域をSPオペランドで表示されているサブプールから解放することを指示. 各記憶域の大きさは, LAオペランドで指示したアドレスから始まるリストの中の値で表示する. また, 各記憶域のアドレスはAオペランドで指定したアドレスから始まるリストの中で表示する.
R	(register)	割当てずみの1個の記憶域をSPオペランドで指定したサブプールから解放することを指示. または, SPオペランドで指定したサブプール全体を解放することを指示.
V	(variable)	割当てずみの1個の記憶域をSPオペランドで指定したサブプールから解放することを指示. 記憶域のアドレスと大きさはAオペランドで指定したアドレスに収容しておく.
LV=	バイト数	解放すべき主記憶域の大きさ（バイト数）を指定.
A=	アドレス	ワード境界から始まる1個以上の連続したフルワードの主記憶域アドレス. EまたはRを指定している場合には解放すべき主記憶域のアドレスを収容した1ワードが必要. Vを指定している場合には2ワードが必要で, 最初のワードには解放すべき記憶域のアドレスを, 2番目のワードには記憶域の大きさを収容していること. Lを指定している場合には, 解放すべき記憶域ごとに1ワードが必要で, そこに記憶域アドレスを入れておくこと. Rの場合には, レジスター1から12のどれかを指定し, そこに主記憶域のアドレスを入れておくこと.
LA=	アドレス	ワード境界から始まる1個以上の連続したフルワードの主記憶域アドレス. 解放すべき記憶域ごとに1ワード必要.
SP=	番号	解放すべき主記憶域のサブプール番号を指定. 0から127であること. 省略するとサブプール0とみなされる.

ので参照されたい.

2)　サブプールの処理

ジョブ・ステップに主記憶域の領域が与えられても, この領域内で利用可能な全記憶域がまだ未割り当ての状態である. サブプール番号を指定した GETMAIN マクロ命令が出されたときにのみ, サブプールが作りあげられる. サブプール番号が指定されないと, 要求した記憶域はサブプール0から確保される. サブプール0だけは, ジョブ・ステップ・タスクが開始されたとき, 監視

プログラムがそのジョブ・ステップ用に作成してくれる.

監視プログラムは, 主記憶域の管理と保護の目的のために, 領域内の主記憶域のすべてを2Kバイト単位のブロックとして取扱う. これらブロックがサブプールに割り当てられるのであり, 主記憶域の要求があると監視プログラムはブロック内のスペースをタスクに割り当てるのである. 指定されたサブプールに割り振られたブロック内の未割当てのスペースが要求された記憶域を満すのに十分であると, そのブロックから主記憶域の要求を出した活動状態のタスクに割り当てる. 逆に, サブプールに割り当てられたブロック内の未割当てのスペースが要求記憶域を満すことができない場合だと, 新規のブロックがサブプールに割り当てられ, そのサブプール内の記憶域が要求を出したタスクに割り当てられる. タスクに割り当てられるサブプールは必ずしも連続した記憶域とは限らない. というのは, 一つの要求で割当てられたブロック群ではないからだ. ともあれ, ジョブ・ステップ用に確保された領域内のブロックだけがサブプールに割当てられるのである.

4個のブロックを含んだ領域の割当ての様子を示したのが**図8-80**である. この図では, 主記憶域のすべての要求がサブプール0からのものであると仮定している. ジョブ・ステップ内のタスクで最初に要求する記憶域は504バイトである. この場合, 図の中のブロックAから満される. 二番目の要求は2,000バイトであり, ブロックAの未使用部分から割り当てるのには大きすぎる. したがって, 監視プログラムはつぎに使用可能状態のブロックであるブロックBを割り振り, このブロックから2,000バイトを活動状態のタスクのために確保する. 3番目の要求は1,000バイトである. これは, ブロックBの残りのスペースで満すことはできない (ブロックは後入れ先出しの順序で検査される). したがって, この1,000バイトの記憶域はブロックAから確保され, 要求タスクに割り当てられる. すべてのタスクがサブプール0を共同しているので, 要求1と3とは同一タスクから出されたものとは限らない. ところで, 要求4は3,000バイトを必要としているので, 連続した未割当てのブロックCとDから要求した記憶域を割り当てることになる. これらのブロックも当然のことながらサブプール0に割り振られる.

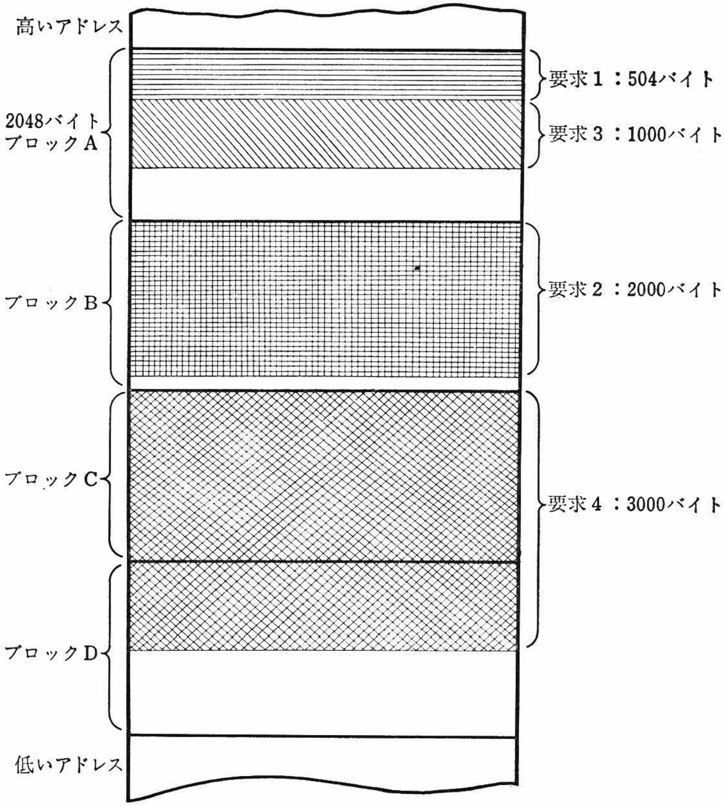


図 8-80　主記憶装置の制御の仕方

図8-80で推察できるように，サブプール0内の単一ブロックが，ジョブ・ステップ内の多くの異ったタスクに割り当てられた沢山の小さな記憶域で構成されることもあり得る．しかも，このような方法で多くのブロックが分割されてしまう可能性もある．ジョブ・ステップ・タスク以外のタスクが確保した記憶域は，タスクが終止しても自動的に解放されない．タスクが終了するまえに，FREEMAIN マクロ命令で小さな記憶域を解放したとしても，それぞれのブロック内に多くの未使用の記憶域が散在する結果となってしまうであろう．一方，監視プログラムは新規の要求に応ずるため次から次へと新しいブロックを割り振ってゆくことになってしまう．これでは困るのである．このような事

態を回避するためには，個々のタスクで別々に使用するサブプールを定義する必要がある．

サブプールは一個のタスクで排他的に使用することもできるし，数個のタスクで共用することもできる．タスクを生成するつど共用すべきサブプールを指示することができる．サブプール0だけは，他のサブプールと異なり特別な指示をしないかぎり，タスクとサブタスクの間で共用される．サブプール0を共用しないときは，監視プログラムがサブタスクで使用する新規のサブプール0を作成してくれる．その結果，タスクとサブタスクの双方でサブプール0からの記憶域を要求したとしても，同一のブロックから双方の記憶域を確保することはない．この場合サブタスクが終了すると，該当タスクのサブプール0内の記憶域は解放される．このサブプールは別のタスクで共用されていないがために，2K単位のブロック全体が再割当て可能な状態に戻されるのである．

サブプール0を共用する必要がある場合でも，個々のタスクで排他的に使用するための別のサブプールを定義することが可能である．サブプール0以外のサブプール内の記憶域を要求すると，監視プログラムは該当サブプール用の新規のブロックを確保し，そのブロックから記憶域を割り当てる仕組みになっている．この結果，該当タスクにサブプールの所有権，つまりそのブロックの所有権が割り当てられる．同一タスクで同じサブプールに追加の要求が出されると，そのブロックと必要なだけ追加のブロックからこの要求を満たす．これと同じサブプールに対して別のタスクからの要求が出されると，監視プログラムは新規サブプール用のブロックを新ためて割り当て，新しいブロックから要求スペースを割り振り，新規サブプールの所有権をそのタスクに与える．

通常のタスクではサブプール番号0から127の番号を指示することができる．FREEMAIN マクロ命令では，サブプール0以外のどのサブプールでも解放することができる．なお，タスクが終止すると，共用していないサブプールは自動的に解放される．明示的要求では0から127までのサブプールしか取扱えないが，**表8-26**でシステムで使用することのできるサブプールをまとめているので必要のつど参考にして欲しい．

表 8-26　要求スペースに対して使用するサブプール番号

サブプール番号	要求対象域	記憶保護キーの割当て	注　釈
246	領域		現存する領域を解放し，新しい領域を割り当てることの要求を示す.
247	領域		新しい領域を割り当てるか，または現存する領域を解放することの要求を示す.
248	領域		ロールアウト／ロールイン・ルーチンで領域の割当てを要求することを示す.
0〜127	領域内のスペース	ジョベ・ステップの記憶保護キー（スペースが解放されると0に戻される）	監視プログラム状態で実行されているプログラムでサブプール 0〜127 を要求すると，サブプール 252 が割り当てられる.
250	領域内のスペース	ジョブ・ステップの記憶保護キー（スペースが解放されると0に戻される）	監視プログラム状態で実行されているプログラムでサブプール 250 を要求すると，サブプール 0 が割り当てられる.
251	領域内のスペース	ジョブ・ステップの記憶保護キー（スペースが解放されると0に戻される）	ジョブ・パック域の再入不可能モジュール
252	領域内のスペース	0の記憶保護キー	ジョブ・パック域の再入可能モジュール
253	システム待合せ域内のスペース	0の記憶保護キー	タスクが終了すると，割り当てられるスペースは解放される.
254	システム待合せ域内のスペース	0の記憶保護キー	ジョブ・ステップが終了すると，割り当てられたスペースは解放される.
255	システム待合せ域内のスペース	0の記憶保護キー	割り当てられたスペースは明示的に解放しなければならない.

①　サブプールの所有と共用

サブプールが作られたときに活動中であったタスクがサブプールの所有権をもっている．しかし，そのサブプールを他のタスクと共用することができるし，場合によってはサブプールの所有権を他のタスクに割り当てることも可能だ．サブプールの処理のためには，2種類のマクロ命令が使える．つまり，

GETMAIN マクロ命令と ATTACH マクロ命令とである. GETMAIN マクロ命令では, サブプール0から127までの記憶域のどれかを要求するために SP オペランドを指定する（表8-22および表8-23を参照のこと）. 一方, ATTACH マクロ命令でサブプールを取扱うオペランドにはつぎのものがある（詳細は第4節の「タスク管理プログラムの機能構造」の表8-1, 表8-2および「サブタスクへのサブプールの割当て」の項を参照のこと）：

- GSPV と GSPL：サブプール0以外のサブプールの所有権を, 新しく生成されたサブタスクに委譲する場合に指定する.
- SHSPV と SHPL：サブプール0以外のサブプールを, 新しく生成されたサブタスと共用する場合に指示する.
- SZERO：サブプール0をサブタスクとの間で共用するかどうかを決定するのに用いる.

これらのオペランドはすべてオプショナルである. 省略されていると, サブタスクにはどのサブプールも与えられず, サブプール0だけが共用されることになる.

② サブプールの生成

新規のサブプールが作られるのは, ATTACH または GETMAIN マクロ命令で前述のオペランドを指定し, しかもそのオペランドで指定したサブプールが活動状態のタスクで所有されていない場合, ないしは共用されていない場合である. ATTACH マクロ命令のオペランドの指定によってサブプールが生成されると, そのサブプール番号は, タスクが所有するサブプールのリストに入れられる. しかし, この時点ではブロックの割当ても記憶域の割当ても行なわれない. 一方, GETMAIN マクロ命令でサブプールが作られると, サブプール番号がブロックに割り振られ, 要求された記憶域が実際にタスクに割り当てられる. いずれの場合にも, サブプールの所有権は活動状態のタスクに与えられることに変りはない. ただし, ATTACH マクロ命令でサブプールを生成すると, 使用したオペランドの種類に応じて, 所有権が委譲されることもそのま

ま保持されることもある．

③　サブプールの所有権の委譲

ATTACH マクロ命令のオペランドで GSPV または GSPL を指定すると，サブプールの所有権を親タスクからサブタスクに移すことができる．サブプールの所有権は，どんなサブタスクにでも委譲できる．関係する2つのタスク間の制御レベルがどうであってもかまわないし，所有権がどのような方法で得られたものであってもよい．ただし，1つ以上のサブタスクの間でサブプールを共有し，そのあとで別のサブタスクに委譲することは許されない．このような場合には，活動状態のタスクが異常終止してしまう．ともあれ，活動状態のタスクが所有権を持っているときだけ，サブプールの所有権を委譲できる．活動状態のタスクがサブプールを共用しており，その所有権をサブタスクに渡そうとすると，まずサブタスクが共用制御権を受取り，そのあとで親タスクはそのサブプールを譲渡する．いったん所有権がサブタスクに渡ってしまうとそれ以降に親タスクで同じサブプール番号を用いると，新規のサブプール が 作 ら れる．サブプールの所有権をもったタスクが終了すると，サブプール内の全記憶域が解放される．それゆえ，サブプールを共用しているタスクやサブタスクがそのサブプールの使用を完了してしまわないと，サブプールの所有権をもったタスクを終了させることはできない．

④　サブプールの共用

サブプールの共同使用権は，そのサブプールの所有権ないしは**共用制御権**（shared control）をもったタスクの サブタスクに 直接与えることができる．ATTACH マクロ命令の SHSPV または SHSPL オペランドを指定することによって，これが達成できるのである．サブプールの所有権ないしは共同制御権をもったタスクは，GETMAIN マクロ命令と FREEMAIN マクロ命令を使ってサブプールの大きさを変更できる．なお，サブプールの共用制御権をもったタスクが終了しても，サブプールは何らの影響もうけない．

⑤　タスク間のコミュニケーションのためのサブプール

主記憶装置の管理という観点からみたサブプールの利点は，個々のサブタスクに別々のサブプールを割り振ることによって，記憶域を小さな区域に細分化してしまうといった事態を防止することにある．別の利点としては，サブプールを用いることによって，タスク間でのコミュニケーションがうまく行なわれるということもある．親タスクでサブプールを作りあげ，このサブプールにサブタスクへ受渡すべきパラメータのすべてを入れておく．サブタスクが ATTACH されると，そのサブプールの所有権をサブタスクに委譲する．サブタスクではすべてのパラメータを処理したあと，FREEMAIN マクロ命令を用いてサブプールの記憶域を解放できる．同様に，親タスク用の2番目のサブプールを作りあげ，サブタスクで処理した結果を収容し，親タスクに戻すこともできる．このためには，サブタスクを生成するときサブプールの所有権をサブタスクとの間で共用するようにしておけばよい．サブタスクを終了するまえに，親タスクに受渡すべきすべてのパラメータをサブプールに入れておく．サブタスクが終了しても，サブプールは共同使用になっているので解放されない．したがって，親タスクでは必要なパラメータを確実に得ることができる．親タスクでは，戻されたパラメータをすべて処理してから，FREEMAIN マクロ命令を与えてそのサブプールを利用可能状態にすればよい．

3） 暗示的要求（implicit request）

主記憶域の暗示的要求がなされるのは，LINK, LOAD, ATTACH および XCTL マクロ命令が出されたときである．これらのマクロ命令はいずれも，ロード・モジュールをローディングするための記憶域を必要としているからである．さらにデータ・セットに対して OPEN マクロ命令を実行した場合にも記憶域の明示的要求がなされる．データ・セットを処理するために必要となるアクセス・ルーチンを主記憶域にローディングする必要があるし，バッファーとして使用する主記憶域も割り当てる必要があるからだ．なお，利用可能なスペースより大きい主記憶域の暗示的要求がなされると，活動状態のタスクは異常終止してしまう．

ここで，暗示的要求で確保した記憶域を解放する場合のやり方と考慮点につ

いてふれておこう.

第五節の「内容監視プログラムの機能構造」でもふれたように，内容監視プログラムは主記憶域に呼び込まれているすべてのロード・モジュールに対して**責任回数**（responsibity count）を示すようにしている．これは，CDE の中のフィールドで表示されており，該当モジュールに対する使用要求の回数を示している．この責任回数は，つぎのような条件で1ずつへらされる：

- ロード・モジュールが LOAD マクロ命令で要求された場合だと，責任回数は DELETE マクロ命令を使ってへらされる.
- ロード・モジュールが LINK, ATTACH ないしは XCTL マクロ命令で要求された場合だと，責任回数は XCTL マクロ命令を使うか，制御プログラムに制御権を戻すことによってへらされる.
- タスクが終了したときには，そのタスクを遂行中に LINK, LOAD, ATTACH および XCTL マクロ命令でロード・モジュールを要求した回数から前記の削除回数を引いた数だけへらされる.

ロード・モジュールで占められている記憶域は，責任回数がゼロに達すると別の使用目的のために利用できる状態になる．ただし，連係パック域に収容されているロード・モジュールに対しては，この原則は適用されない．ともあれ，プログラム設計にあたっては，プログラムの実行時間と主記憶域の効率的利用との間の最適なバランスを考慮に入れてモジュールの設計をすることが要求される.

当然のことながら，ジョブ・ステップの実行中にロード・モジュールを何回もくり返して使用する場合には，LOAD マクロ命令でロード・モジュールを主記憶域に呼び込み，その使用が完了したらはじめて DELETE マクロ命令を出すようにした方がよい．この場合，該当モジュールを常時主記憶域に収容しておいた方が，必要のつど呼び込むよりはるかに効率的であるからだ．逆に，ジョブ・ステップの実行中に1回しか利用しないとか，その使用が時間的に分散しているような場合には，LINK マクロ命令でモジュールをローディングし，その使用が完了したときそのモジュールから XCTL マクロ命令を出すか

制御プログラムに制御権を戻す方法をとると，主記憶域が節約できる．

DCB を含んだロード・モジュールを削除するさいには若干の問題が残る．DCB の使用に先立って OPEN マクロ命令を与え，使用後には CLOSE マクロ命令を与える必要があることはいうまでもない．DCB に対して CLOSE マクロ命令が与えられないと，タスクが終了したときに制御プログラムがCLOSE マクロ命令を出してくれる．ところで，DCB を収容しているロード・モジュールがすでに主記憶域から除去されてしまっているとすると，CLOSE マクロ命令を実行しようとすることによってタスクの異常終止が生じてしまう．したがって，ロード・モジュールを削除するまえにCLOSE マクロ命令を与えるか，タスクの終了時にも DCB が主記憶域に残っているように配慮する必要がある．

2. 主記憶域割当ての概観

記憶域の要求のすべてが GETMAIN ルーチンで処理されることは前にふれた通りである．これには，領域の要求，領域内のスペースの要求，およびシステム待合せ域内のスペースの要求が含まれる．基本的には，GETMAINルーチンが，利用可能状態のスペースを表示する要素からなる待行列を探索し，要求タイプのスペースを満す記憶域をつきとめる．必要なスペースがみつかると，GETMAIN ルーチンは関連する待行列を更新し，そのスペースがこれ以降別の目的に利用できないことを示すと同時に，要求プログラムに確保したスペースのアドレスを戻す．要求スペースが確保できない場合には，GETMAIN ルーチンは要求された記憶域のタイプ，つまり新規領域の要求，現存する領域内のスペースの要求，あるいはシステム待合せ域内のスペースの要求に応じた処理をおこなう．

新規領域の要求スペースが利用可能状態になく，しかも要求が条件つきである場合だと，GETMAIN ルーチンは戻りコードを準備し，タイプⅠエグジット・ルーチンを経由して制御権を要求プログラムに渡す．一方，無条件要求であると，GETMAIN ルーチンは，動的区域内のスペースが十分利用できるようになるまで要求プログラムのタスクをディスパッチ不能の状態におく．その

あとで，作動可能状態のタスクのうちで最も優先順位の高いタスクに制御権を受け渡す．

現存する領域内の要求スペースが利用可能状態になく，しかも要求が条件つきである場合だと，GETMAIN ルーチンは戻りコードを準備し，タイプIエグジット・ルーチンを経由して制御権を要求プログラムに戻す．これは，新規領域のスペースを要求した場合と全く同じ処理の仕方である．一方，無条件要求であると，GETMAIN ルーチンは解放可能なスペースを見出し，要求タスクに割り当てようとする．まず，要求タスクの領域内にあるモジュールのうち除去しても差支えない未使用のモジュールを捜す．未使用のモジュールを除去しても，十分なスペースを使用することができず，しかもロールアウト／ロールイン機能が適用できない場合には，要求プログラムのタスクを異常終止させることになる．ところが，ロールアウト機能がシステムに組み込まれており，要求プログラムのタスクにロールアウトをひき起す資格が与えられている場合には（第7章「ジョブ管理の諸機能とその構造」の第四節「ジョブ制御ステートメントの構造」のうちの8）ロールアウトの指定を参照のこと），GETMAIN ルーチンは**ロールアウト／ロールイン・モジュール**（rollout/rollin module）との連係をスケジュールする．ロールアウト／ロールイン・モジュールは，要求プログラムのジョブ・ステップで使用するための追加領域の1時的な割り振りを行なおうと試みる．追加領域は動的区域の空きのスペースから得るか，他のジョブのジョブ・ステップにすでに割り当てられている領域を一時的に再割当てすることによって得ようとする．ロールアウト／ロールイン・モジュールで必要な領域を見出せないと，要求プログラムのジョブ・ステップを異常終止させるか，必要な領域が利用可能になるまでそのジョブ・ステップをディスパッチ不能状態におくことになる．どちらが選択されるかは，ユーザーの作成した附加ルーチンでのオプションにかかっている．

3番目のケースは，システム待合せ域のスペースを要求する場合である．この場合，システム待合せ域に利用可能なスペースがないと，GETMAINルーチンはシステム待合せ域内の CDE を除去して，CDE 用スペースを解放する．これでも十分な区域が得られないと，システム待合せ域を拡張しようと試み

る．システム待合せ域に隣接する未使用の動的区域を２Ｋブロックの単位でシステム待合せ域に割り振ることによって，これを実現しようとするのである．隣接する動的区域がすでに領域に割り当てられていてシステム待合せ域が拡張できないとか，システム待合せ域が拡張できても要求を満足させることができないとなると，システム待合せ域内の144バイトの大きさが利用できないかどうか判定する．これが不可能な場合には，GETMAIN ルーチンは要求タスクの TCB に特別な完了コードを入れ，TCB をディスパッチ不能にしたあとで，CPU を待ち状態においてしまう．144バイトの大きさが利用可能だと，ABRTEM ルーチンを使って要求タスクを異常終了させるようにスケジュールする．144バイトは，要求プログラムを終了させる ABEND ルーチンのための SVRB を作りあげるのに必要となるのである．

以上みてきたように，主記憶装置割当て管理の対象となるのはつぎの３種類であり，それぞれ管理の仕方が異なる：

- 動的区域からの領域の割当て
- 領域内のスペースの割当て
- システム待合せ域のスペースの割当て

さらに，領域内のスペースの割当てに深い関係をもつものとしてロールアウト／ロールインの機能がある．そこで，これら４つの項目について主記憶装置監視プログラムがどのように機能するかを構造的に説明してゆこう．

3. 動的区域からの領域の割当てと解放機能

領域用のスペースは，主記憶装置の動的区域から確保される．すでに第４節の「主記憶装置の監視機能」の個所で概観したように，動的区域内のスペースのうち領域に割り当てられた部分といまだ領域に割り振られていない部分との管理は，**フリー・ブロック待行列要素**（Free Block Queue Element；略して **FBQE**）を用いて行なわれる．FBQE は，動的区域のスペースのうちいまだ領域に割り当てられていない区域のそれぞれの下位アドレスの場所に12バイトの大きさで作られる（図 8-13 を参照のこと）．この FBQE の中の SIZE フィー

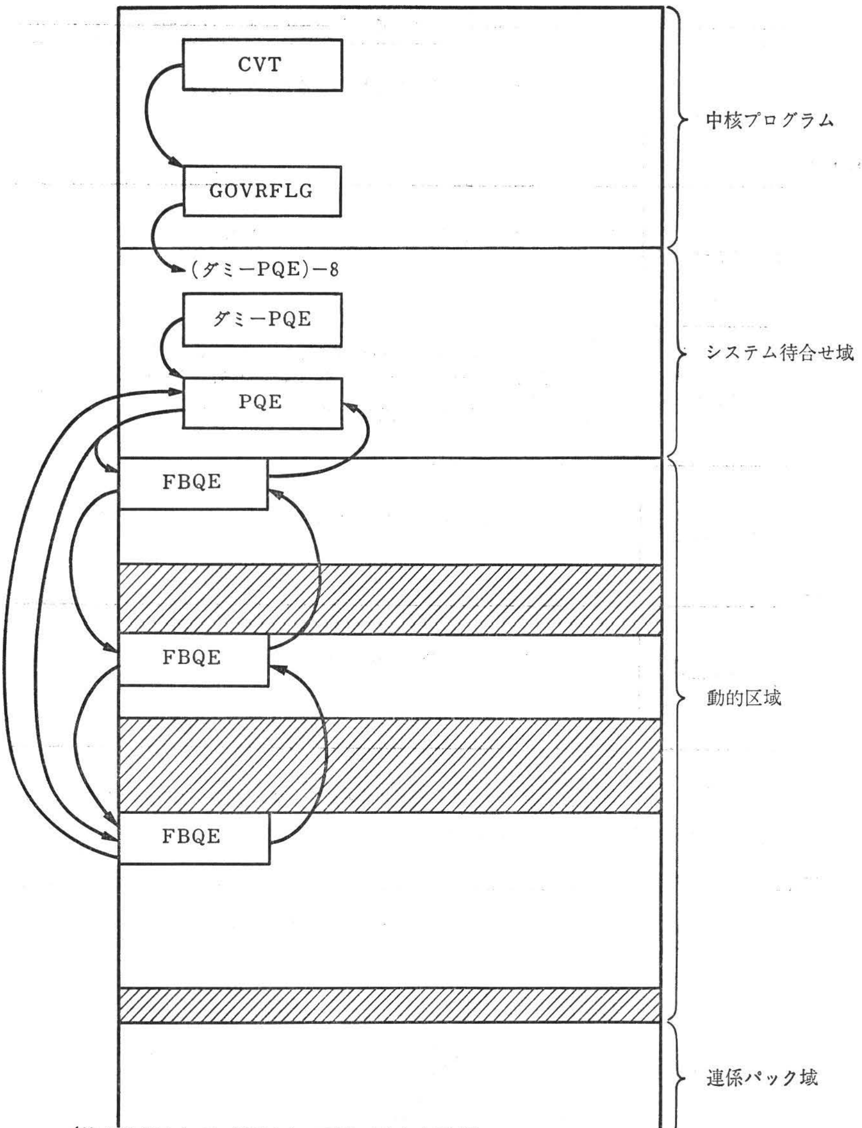
バイト			
0	予約フィールド	1	FWDPTR：領域内のつぎの高位アドレスの FBQE へのポインター．最高位アドレスにある FBQE には，PQE のアドレスが入っている．
4	予約フィールド	5	BCKPTR：領域内のつぎの下位アドレスのFBQEへのポインター．最下位アドレスにある FBQE には，PQE のアドレスが入っている．
8	予約フィールド	9	SIZE：2K ブロック・セット内のバイト数

図 8-81　フリー・ブロック待行列要素（FBQE）の形式と内容

ルドには空いている区域の大きさが示されている．FBQE は，お互に前方向と後方向から連鎖されている．したがって，主記憶装置監視プログラムは，高位のアドレスから空きの区域を探索することも，逆に低位のアドレスから探索することもできる（**図 8-81** および**図 8-82** を参照のこと）．

このように FBQE はお互に連鎖されているのであるが，FBQE で構成される待行列は，システム待合せ域（SQA）の**区分待行列要素**（Partition Queue Element；略して **PQE**）からポイントされている．この場合の PQE は動的区的全体を表わす PQE である．動的区域内の空きの場所を管理するための制御ブロックの連鎖関係を示したのが**図 8-32** である．

GOVRFLB（**図 8-83** 参照）は，CVT からポイントされていると同時に PQEPTRフィールドで**ダミー区分待行列要素**（Dummy Partition Queue Element；略して **DPQE**）をポイントしている．実際には，PQEPTR フィールドには，DPQE から 8 だけ引いたアドレスが収容されているのであるが，これをもとに DPQE をつきとめる仕組みとなっている．**図 8-84** で示しているように，PQE は2 ワードから構成されている．最初のワードには，まだ領域に割り当てられていない空きの区域を記述している PQE のアドレスが収容される．一方，2番目のワードは，NIP で作りあげられた最後の PQE のアドレスを持っている．なお，図 8-82 で図示しているように，PQE はその PQE で記述している記憶域に関連する FBQE 待行列の最初と最後をポイントしている．PQE の形



(注：斜線はすでに領域として割当てられた区域)

図 8-82　制御ブロックの連鎖構造

←4　バ　イ　ト→

バイト		
0	予約フィールド	1　SQBOUND：SQAを越えた最初のバイトのアドレス
4	予約フィールド	5　DQESQES：SQA を記述したDQEのアドレス
8	予約フィールド	9　PQEPTR：ダミーPQEから8引いたアドレス．ダミーPQE は未割り当ての記憶域を記述するPQEをポイントしている
12	予約フィールド	13　SZDPRS：NIP後のハイラルキー0で使用可能な記憶域の大きさ
16	予約フィールド	17　SZDLCS：NIP後のハイラルキー1で使用可能な記憶域の大きさ
20	COUNT：“VARY STORAGE, OFFLINE” 指令の個数	21　VQEPTR：多重プロセシング・システムで除去すべきためにスケジュールされた記憶域を記述した最初のVQEのアドレス

図 8-83　主記憶装置待行列のための起点リスト (GOVRFLB) の形式

0　チェイン内の最初のPQEのアドレス	4　チェイン内の最後のPQEのアドレス

図 8-84　ダミー区分待行列要素 (DPQE) の形式

式は，図 8-85 に示した通りである．

ともあれ，領域の割当てと解放は，FBQE の内容の更新，FBQE の追加と削

オフセット				
0	PQEFFBQE：当該 PQE で記述した領域内の最初の FBQE のアドレス．FBQE が全く存在しないと，PQE アドレスが収容される．			
4	PQEBFBQE：当該 PQE で記述した領域内の最後の FBQE のアドレス．FBQE が全く存在しないと，PQE アドレスが収容される．			
8	PQEFPQE：待行列上のつぎの PQE のアドレス．最後の PQE では0が入っている．			
12	PQEBPQE：待行列上の先行 PQE のアドレス．最初の PQE では0が入っている．			
16	PQETCB：そのスペースが属するジョブ・ステップ用 TCB のアドレス．そのスペースが未割り当てのフリー・スペースから得られたときには0が入っている．			
20	PQESIZE：当該 PQE で記述されている領域のサイズ（常に2048の倍数）			
24	PQEREGN：当該 PQE で記述されている領域の最初のバイトのアドレス			
28	PQERFLGS：ロールアウト標識	29 PQEHRID：ハイラルキー標識	30 予約フィルード	31 予約フィールド

図 8-85　区分待行列要素（PQE）の形式

除を行い，必要に応じて制御ブロック間の連鎖関係を更新することによって達成されるのである．以下で領域の割当てと解放について具体的に説明することにしよう．

1）領域の割当て

領域を割り当てるために，GETMAIN ルーチンは動的区域内の FBQE を高いアドレスから捜してゆく．要求を満すに十分な大きさを持った区域に対応する FBQE が見つかると，つぎの手順で領域の開始アドレスを算出する：

開始アドレス ＝ 空いている記憶域のサイズ － 要求領域のサイズ ＋ 空きの記憶域を表示している FBQE のアドレス

そのあとで，空きの記憶域を表示している FBQE 内のバイト数（SIZE フィ

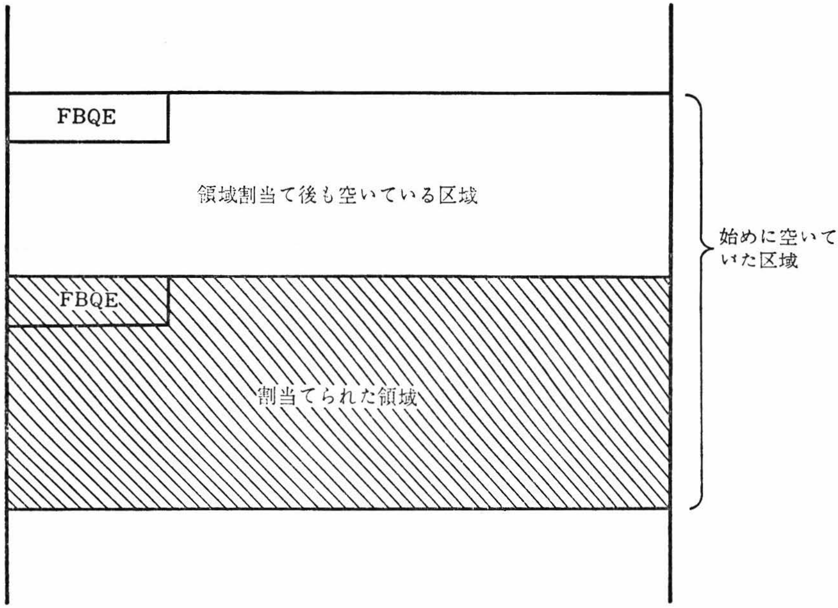


図 8-86　領域割り当ての様子

ールド）から新規の領域用に確保したバイト数を引くことによって，FBQE の内容を更新する．また，新規の領域用の FBQE も作りあげる．この結果，領域の割当てを行なったあとの主記憶域は**図 8-86** で示したようになる．さらに，新規に確保した領域に対するポインターを作りあげる（**図 8-87** 参照のこと）．

実際には，GETMAIN ルーチンは各領域に対応してつぎの 3 種類の制御ブロックを作りあげ，相互に連鎖関係を作るのである．つまり，領域の最初には FBQE を，システム待合せ域には DPQE と PQE とをそれぞれ 1 個ずつ作りあげる．そして，領域用に確保した記憶域のバイト数を FBQE の SIZE フィールドに入れる．また，DPQE では PQE をポイントするように，一方，PQE は FBQE をポイントするようにそれぞれの所定フィールドに該当アドレスを挿入する．さらに，GETMAIN ルーチンは PQE の所定フィールドに領域のサイズをセットする．最後に，DPQE から 8 だけ引いたアドレスを領域を要求したジョブ・ステップ・タスク用の TCB の TCBPQE フィールドに挿入することによって，領域の割当てに関する処理が完了する．

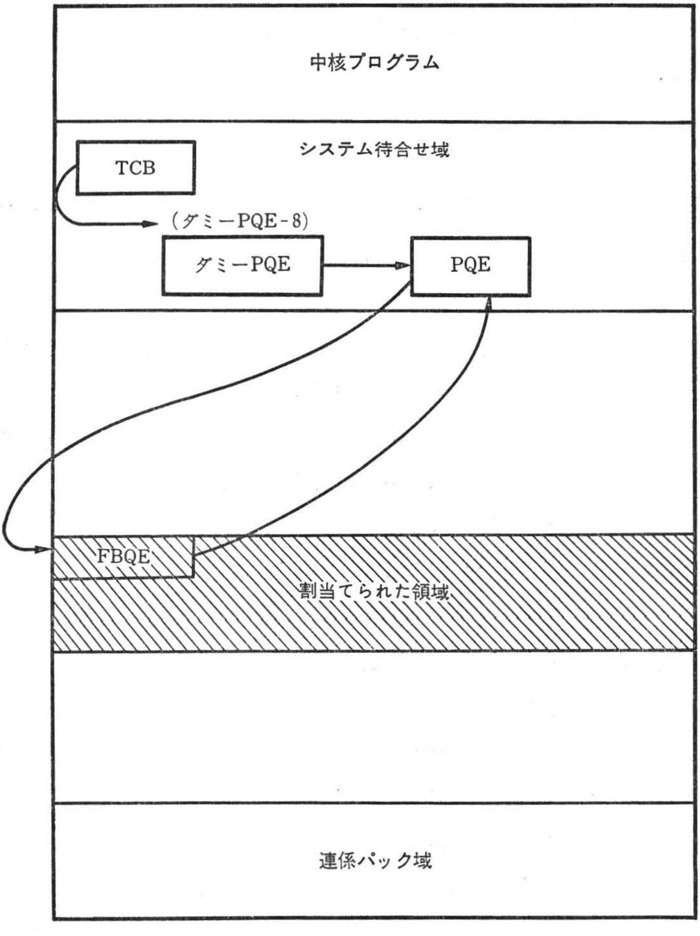


図 8-87　新しく確保した領域に対するポインター

一方，動的区域の中に要求領域を満足させるだけの十分な空きの区域がない場合には，要求が条件つきか無条件要求かによって処理の仕方が異なる．条件つき要求だと，GETMAIN ルーチンではレジスター15に戻りコード4をセットし，要求スペースの割当てが出来なかった旨を知らせる．そのあと，タイプⅠエグジット・ルーチンを経由して要求プログラムに制御権を戻す．無条件要求だと，TCB の TCBNDSP 1 フィールドにビットを立て，タスクのディスパ

ッチを一時的に不能にし，領域の再要求に備えることができるようにしておく．そのあとで，目下もっとも優先順位の高い作動可能状態にあるタスクに制御権を渡す．

2) 領域の解放

領域の解放は，FREEMAIN ルーチンで行なう．FREEMAIN ルーチンは空きの区域を示す FBQE を作りあげるか，あるいは FBQE の内容を更新することによって，解放すべき領域を再使用可能な状態にする．

領域を解放するために，その領域を使用していたタスクを表示しているTCBの TCBPQE フィールドをもとに PQE のアドレスをつきとめ，PQE が占めているスペースを解放する．さらに，PQE でポイントされている領域を解放する．領域の置かれている位置によって，その解放の仕方には差がある．この様子を示したのが**図 8-88** である．

解放すべき領域が空いている区域と隣接していない場合には，FREEMAIN ルーチンはそのスペース用の FBQE を作りあげ，領域として割当て可能なスペースを表示している一連の FBQE と連鎖する．

解放すべき領域が空いている区域と隣接している場合には，その区域と結合される．この場合には，解放すべき領域のバイト数を空いている区域用の FBQE の SIZE フィールドに加算することによって，FBQE の内容を更新する．新規に FBQE を作ることはしない．必要なら，新しく拡張された区域の先頭に FBQE を再配置してくれる（図 8-88 の 3 ）の場合）．また，上下に空きの区域が隣接しているときには，空きの区域が拡張されることによって不要となった FBQE は連鎖から削除されてしまう（図 8-88 の 4 ）の場合）．

解放すべき領域が要求タスクの所有でない場合，つまりロールアウトによって獲得した領域である場合には，該当領域を所有しているジョブ・ステップをロールインすることができる．この場合には，FREEMAIN ルーチンは FREBRF ルーチンにブランチする．FREBRF ルーチンでは領域の属性をテストし，支障がないことが分ると，現行のジョブ・ステップから領域を解放し，ロールアウト／ロールイン・モジュールの連係を準備する．

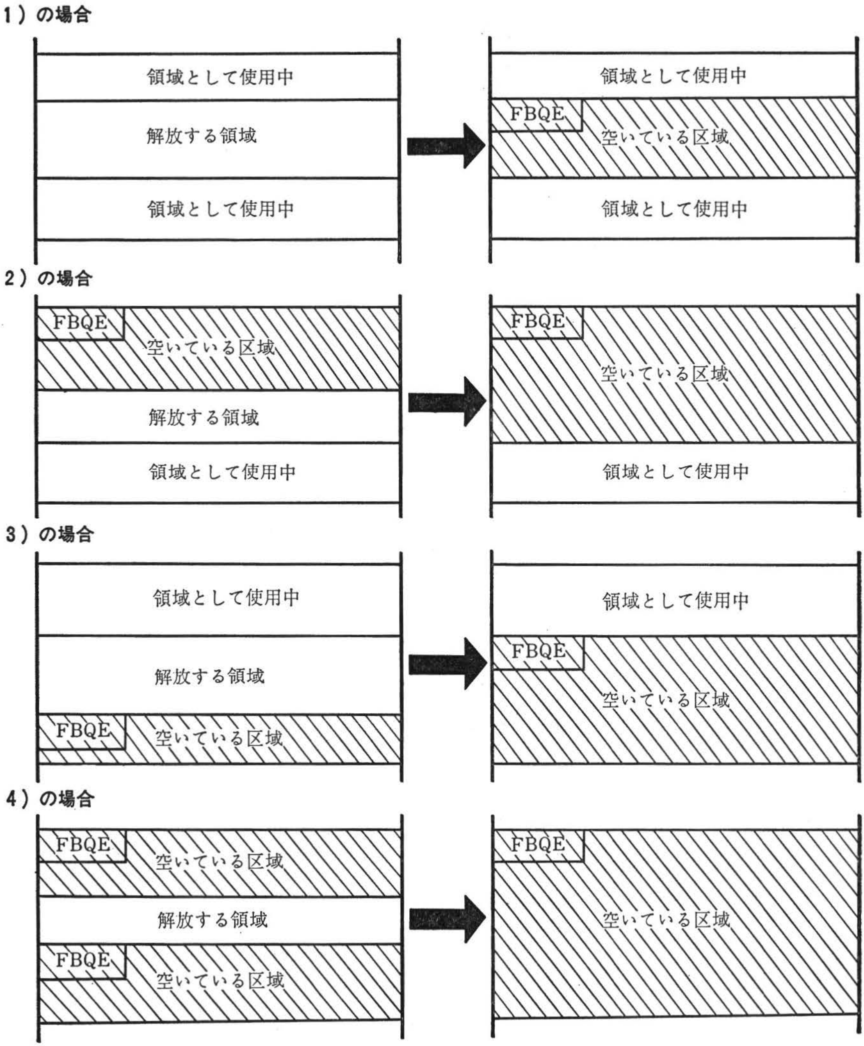
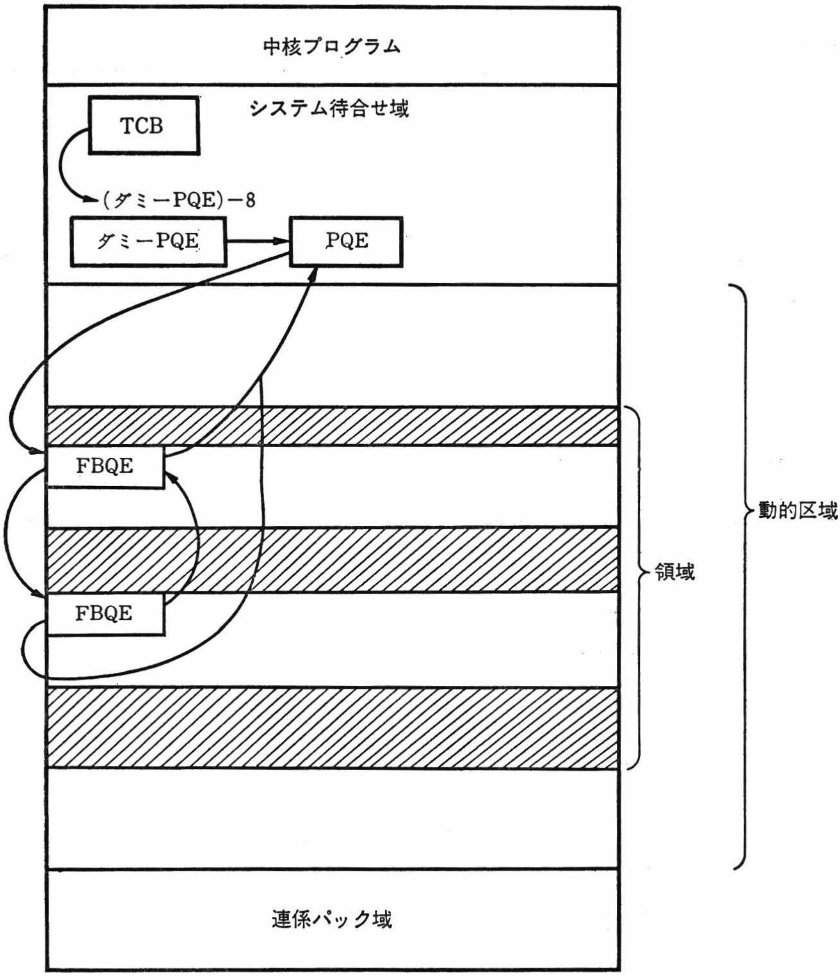


図 8-88　領域の解放の様子



(注：斜線の部分は，領域内ですでに使用中の区域を示す)

図 8-89　領域内の空スペースの管理の仕方

4.　領域内のスペースの割当てと解放機能

領域内の空いている区域の管理の仕方は，動的区域内の空いている区域を管理する場合と同じように FBQE を使って行なう．領域内の空いている区域の一番低いアドレスに FBQE が作られ，**図 8-89** に示しているようにチェーニン

グされる. DPQE, PQE および FBQE の形式と役割りとは, 動的区域の場合のそれと全く同じである. 異なるのは, 動的区域の空きのスペースではなく, 領域内の空きのスペースの管理に用いられる点である. なお, DPQE と PQE とは領域が割り当てられた時点に作りあげられたものである. 一方, FBQE は領域が割り当てられた時点では領域全体が未使用の状態であることから, 唯1個の FBQE だけが作りあげられる (図 8-87 を参照のこと). しかし, ジョブ・ステップの実行が開始されると種々のスペース要求がなされ, 領域内にはいくつかの空きの区域が存在することになり, それぞれに対応して FBQE が作りあげられる.

一方, 領域内の使用中の区域にはサブプール番号がつく. 同一サブプール番号をもった区域が2つ以上存在することも許される. 各サブプールに応じて1個の**サブプール待行列要素** (Subpool Queue Element; 略して **SPQE**) が作られる. また, サブプール番号のついた区域ごとに**ディスクリプター待行列要素** (Descriptor Queue Element; 略して **DQE**) が作られる. SPQE と DQE のいずれもシステム待合せ域に収容される. なお, 同じサブプール番号を持つ DQE はチェーニングされており, 最初の DQE は SPQE からポイントされている.

図 8-90 をみて欲しい. サブプールは 2K のブロックを単位として割り当てられる. あるタスクでプログラムAのためにサブプール 252 を要求したとする (①). 前にもふれたように, サブプール252は再入可能ルーチンを収容するとか, 記憶保護を必要とする制御プログラム用のデータを収容する記憶域のいずれかに適用される. この例では, 再入可能なプログラムであるプログラムAを収容するための区域である. なお, サブプール 252 に割り当てられる記憶域は領域の上位アドレスから確保されてゆく. ともあれ, サブプール 252 のための SPQE が作られ, 確保されたサブプールの区域をポイントする DQE が作成される. プログラムAの実行中に GETMAIN マクロ命令でサブプール3を要求したとする (②). この結果, サブプール3のための SPQE が作りあげられると同時に, 割り当てられたサブプールをポイントするための DQE が作成される. さらに, プログラムBを収容するために再びサブプール 252 を要求した

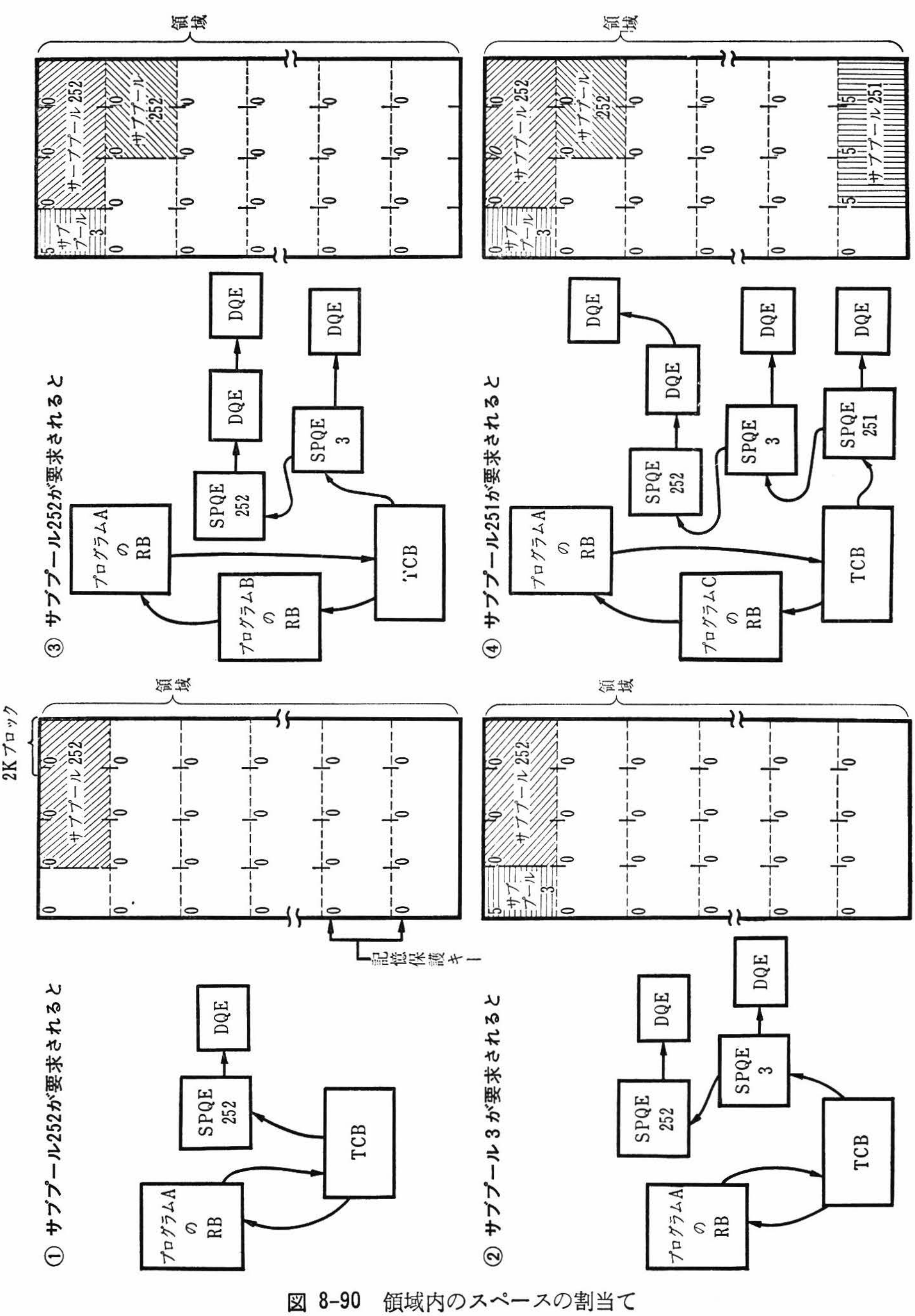


図 8-90　領域内のスペースの割当て

	ビット 0 ： 1 ： 2		
バイト 0	標識（注）	（予約フィールド）	1　SPQEPTR：つぎの SPQE へのポインター．ゼロだと，当該要素が待行列上の最後の SPQE であることを示す．
4	SPID：サブプール番号		5　DQEPTR：当該サブプールの最初の DQE へのポインター．サブプールが共用されていると，所有権をもった SPQE を指示する．

注： 標識（バイト 0）
ビット 0：ゼロだとサブプールが当該タスクに属することを示す．1だとサブプールが共用されていることを示す．
ビット 1：通常ゼロ．1だと当該 SPQE が待行列の最後の要素であることを示す．
ビット 2：1だと，当該 SPQE が他のタスクと共用されていることを示す．

図 8-91　サブプール待行列要素（SPQE）の形式と内容

とする（③）．すでにサブプール 252 用の SPQE が存在しているので，新規に SPQE を作る必要はない．しかし，サブプールの区域としては別の記憶域が割り当てられたので，そのための DQE が作られる．新しく作られた DQE は先行する DQE からポイントされる．このあとで，ユーザー・プログラムであるプログラムＣのためのサブプール 251 が要求されたとする（③）．サブプール 251 は領域内の下位アドレスから確保されてゆく．サブプール 251 用の区域はそれまで存在していなかったので，SPQE と DQE の双方が作りあげられる．

SPQE の形式は**図 8-91** に示している．2 ワードの大きさであり，つぎの SPQE へのポインターと最初の DQE へのポインターを収容するフィールドがある．また，当然のことであるがサブプール番号をセットするフィールドも用意されている．一方，DQE の形式と内容とを示したのが**図 8-92** である．後述する**フリー待行列要素**（Free Queue Element；略して **FQE**）へのポインターと後続する DQE のポインターを収容するフィールドがある．また，割り当てられたサブプール区域のアドレスとその区域の大きさを入れるフィールドとがある．

ところで，サブプール番号のついた区域は 2 K のブロック単位でとられるので，必ずしも区域全体が使用されているとは限らない．そこで，サブプール番号のついた区域内の空いているスペースの一番低いアドレスに FQE が作られ

バイト			
0	(予約フィールド)	1	FQEPTR：最初のFQEへのポインター．全ブロックが割り当てられたとき（つまり，FQEが存在しないとき）はゼロ．
4	(予約フィールド)	5	DQEPTR：つぎのDQEへのポインター．最後のDQEの場合にはゼロ．
8	DQEHRID：標識	9	ブロック・アドレス：当該 DQE で記述されている最初の2Kブロックのアドレス
12	(予約フィールド)	13	長さ：当該 DQE で記述されている区域の大きさ（バイト数）

図 8-92　ディスクリプター待行列要素（DQE）の形式と内容

る（**図 8-93** を参照のこと）．図でも分るように，同一区域内の FQE はチェーニングされている．一方，最初の FQE はDQE からポイントされている．FQE は，**図 8-94** で示しているように，同一区域内の下位アドレスにある空きのスペースの FQE へのポインターを収容するフィールドと，空きのスペースの大きさを示すフィールドとで構成されている．

1)　領域内のスペースの割当て

領域内のスペース要求は GETMAIN マクロ命令を通じて行なわれる．すでに説明したように GETMAIN マクロ命令ではサブプール番号が指定できる．ともあれ，要求されたスペースが利用可能状態にあるかどうかによって，GETMAIN ルーチンでの処理の仕方は大きく異ってしまう．そこで，要求スペースが利用可能である場合と，そうでない場合に分けて処理の仕方を説明することにする．

①　要求スペースが利用可能な場合の処理

GETMAIN ルーチンは，サブプールに対する最初の要求を受けとると，システム待合せ域に SPQE を作りあげる．さきにみたように，SPQE にはサブプール番号が収容されている．すでに別のサブプールが存在しているときには，そのサブプールの SPQE があるので，それをポイントするアドレスを SPQE の所定のフィールドへ入れてくれる．GETMAIN ルーチンは，要求を受け

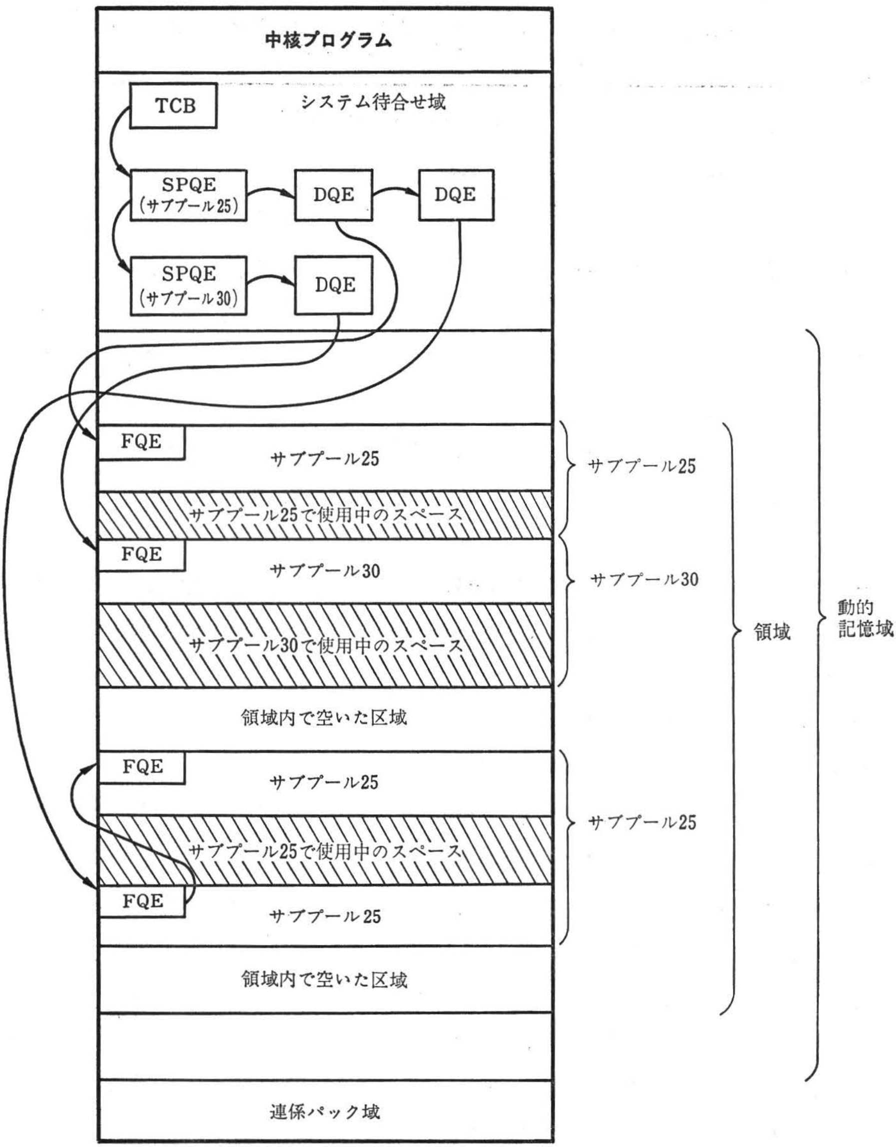


図 8-93 サブプール内の空スペースの管理の仕方

バイト		
0	(予約フィールド)	1 FQEPTR：つぎにつづく下位アドレスのフリー・スペースへのポインター
4	(予約フィー　ド)	5 長さ：フリー・スペースのバイト数

図 8-94　フリー待行列要素 (FQE) の形式と内容

るつど SPQE のポインターをもとに各々の SPQE を探索してゆき，要求されたサブプールがすでに存在しているかどうか確めるのである．領域内に該当するサブプール番号を持った区域が存在していないことが確認できると，上前のように SPQE を作成する．

さらに，GETMAIN ルーチンはシステム待合せ域に DQE を作りあげ，その DQE のアドレスを SPQE の所定の場所に挿入する．DQE には，サブプール内のブロックに割り振られた記憶域のバイト数が収容される．なお，サブプールに割り当てられる領域内のスペースは2Kブロックの倍数を単位としていることは前にふれた通りである．

すでに存在する DQE で記述されているスペースでまかないきれないようなスペースの要求が同一サブプールに対して出されると，追加の区域がそのサブプールに割り当てられる．このためのスペースは FBQE で表示されている空きの記憶域から確保される．確保された新しい区域に対しては DQE だけが作られる．その区域に割り当てられたバイト数が，確保された元の記憶域の FBQE の所定のフィールドから差し引かれる．また，必要に応じて FBQE は再配置される．これによって，FBQE は新たな空きのスペースを反映した内容に更新されるのである．ともあれ，同一サブプール内のスペースを表示しているすべての DQE はお互に連鎖される．2Kブロック単位のスペースが割り当てられると，ブロック単位に記憶保護キーがセットされる (図 8-90 参照)．この記憶保護キーはブロックが解放されるとゼロに戻される．

DQE で表示されている区域に空きのスペースがあると，そのスペースの最初に FQE を作りあげ，空きのスペースの大きさを示すバイト数を入れる．なお，一つの区域内の FQE はすべてお互に連鎖される．GETMAIN ルーチン

は，最初の FQE のアドレスを DQE の所定のフィールドに挿入する.

領域内にすでに同一サブプール番号の区域がある場合だと，現存するサブプール内の空きのスペースをつきとめるために，GETMAIN ルーチンは SPQE を順番に探索し，まず該当サブプールの SPQE をみつけ出す．それをもとに，最初の DQE のアドレスを決定し，要求スペースを満すに足る空きのスペースをもった FQE がみつかるまで DQE を順番に捜してゆく．十分なスペースが存在すると，そこからスペースを割り当て，FQE 内の空きのスペースを示しているフィールドの値を更新する．一方，そのサブプールに対応する区域を全部捜しても，十分な大きさの空きスペースがない場合には，まだどのサブプールにも割り当てられていないスペースをつきとめ（これは FBQE を捜索して行なうのだが)，要求されたサブプールに追加する．この場合には，SPQE はすでに存在するので，DQE だけを作りあげ，対応するサブプールの DQE チェーンに加える.

このようにして得られたスペースのアドレスは，レジスター1ないしは指示された記憶域にセットされ，要求プログラムに返される.

② 要求スペースが利用可能でない場合の処理

領域内に要求を満すに十分な空きのスペースが存在しないことが判明すると，GETMAIN ルーチンは，つぎのような処理を行なうことによって探索範囲を拡張する：

- 領域内に存在するモジュールのうち，使用されていないものを追放する.
- ロールアウトを通じて該当ジョブ・ステップで借りた領域内に空いたスペースがあるかどうか調べる.
- 追加の領域を借りるために，ロールアウト／ロールイン・モジュールを連係できるかどうか調べる.

領域内の使用されていないモジュールを追放するためには，ジョブ・パック域（サブプール251と252）内のすべての CDE を調べる．解放標識が立っている CDE がみつかると，その CDE を除去し，CDE，エクステント・リストお

よび該当モジュールで占められていたスペースを解放する．追放可能なすべてのモジュールに関するスペースを解放した結果，要求スペースを満すに足りる空きのスペースが得られた場合には，そこから要求スペースを割り当てる．なお，追放可能なモジュールの例としては，再入可能なモジュールですでにその使用が終了し，目下使用要求がないモジュールがこれに相当する．

モジュール追放後も要求スペースに応ずるだけの空きスペースが得られない時には，ロールアウトを通じて他から借りた領域内の空きのスペースを調べる．その領域内に十分な大きさの空いたスペースがある場合には，そこから要求されたスペースを割り当てる．その領域内に十分なサイズの空きスペースがないとか，ロールアウトで領域を借りていない場合にはつぎのように処理する：

- **条件つき要求の場合：** 戻りコード4をセットしたうえで，タイプIエグジット・ルーチンを経由して要求プログラムに制御権を戻す．
- **無条件要求の場合：** ロールアウト機能がシステムに組み込まれていないときは，要求プログラムのタスクを異常終止させる．逆に，ロールアウト機能が組み込まれていると，ロールアウトの可能性を調べる．

つぎの条件が全部満足されていると，GETMAIN ルーチンからロールアウト／ロールイン・モジュールに制御権が渡される：

- 無条件要求である．
- ロールアウト機能がシステムに組み込まれている．
- スペース要求が問題プログラム，または問題プログラムのもとで働くシステム・ルーチンから出されている．
- スペース要求を出したタスクがロールアウトできるジョブ・ステップに属していること（これは，JOB または EXEC ステートメントの ROLL パラメータで指定される）．

なお，ロールアウト／ロールイン・モジュールの機能に関しては「6．ロールアウト／ロールインの処理機能」の項で詳述する．

2） 領域内のスペースの解放

領域内のスペースの解放も FREEMAIN ルーチンで行なう．FREEMAIN マクロ命令のオペランドで指定したサブプール番号にもとづいて，それに対応した SPQE をみつけ出し，SPQE でポイントされている DQE を順番に探索し，解放しようとしているスペースを含んだ区域を表示している DQE をみつける．

解放しようとしているスペースがその区域内の空きのスペースに隣接している場合だと，空きのスペースが拡張されるだけであるから，FQE 内の長さフィールドを更新するだけでよい（図 8-95 の 1）を参照）．場合によっては，FQE を再配置する必要が生じる．これは，解放しようとしているスペースが空きのスペースよりも下位アドレスに存在している時に生じる．

解放しようとしているスペースが，その区域内の空きのスペースに接していないときには，新しく FQE を作りあげ，FQE チェインに追加する（図 8-95 の 2）を参照）．

解放されたサブプール内のスペースが2048バイト境界から始まる 2 K バイト・ブロックを含む場合は，そのスペースを領域内の空きの区域に返す（図 8-95 の 3）を参照）．

なお，解放したスペースが，ロールアウトによって得た領域内のスペースである場合には，その領域内のすべてのスペースが解放されているかどうか調べ，すべてのスペースが解放されていることが確認されるとロールインを行なう．

5． システム待合せ域のスペースの割当てと解放機能

システム待合せ域は，システムが各種プログラムを制御するための制御ブロックなどを収容するのに使用される．したがって，システム待合せ域の使用は制御プログラム・ルーチンだけに限定されており，ゼロの記憶保護キーのもとで実行されるルーチンだけがこの区域内のスペースを利用することができる．

システム待合せ域のスペースを得るためには，GETMAIN マクロ命令でサブプール 253，254，または 255 のいずれかを指定する必要がある．各サブプー

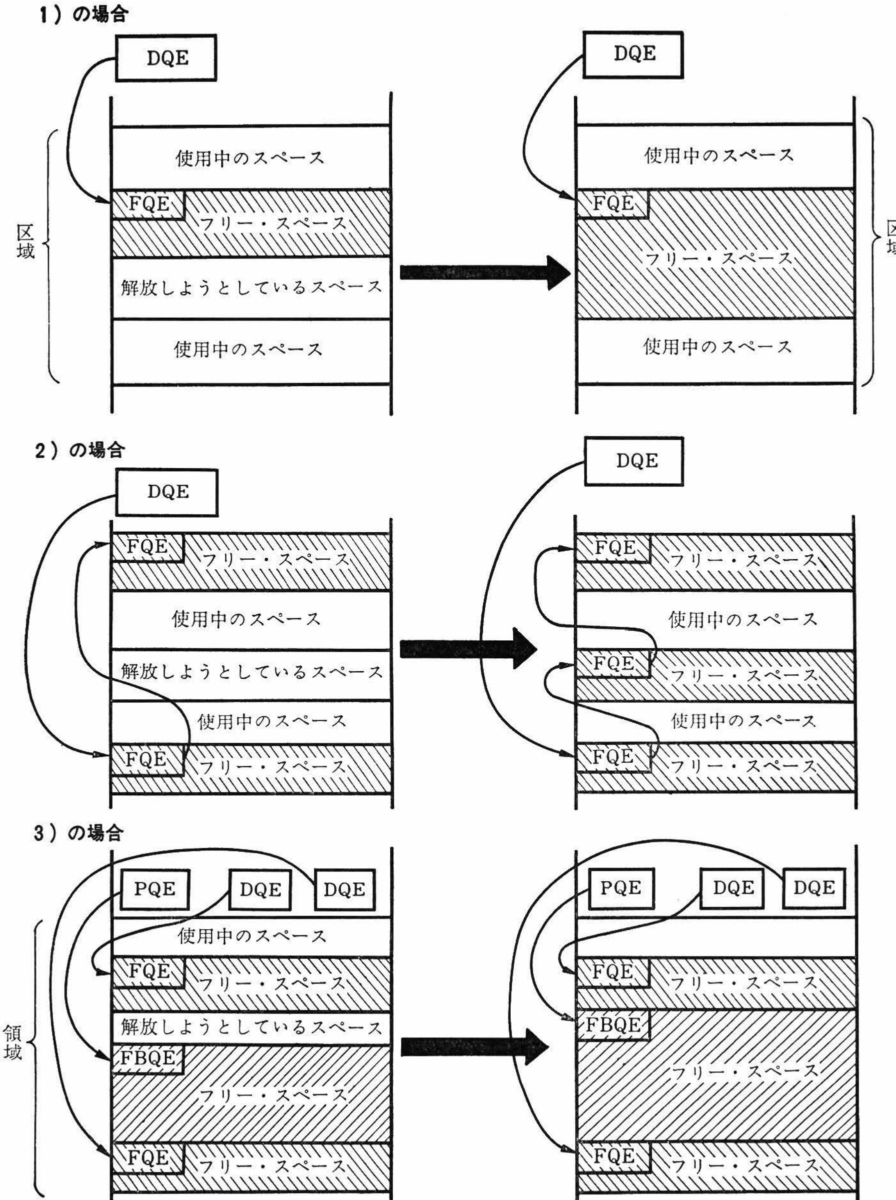


図 8-95　領域内のスペースの解放

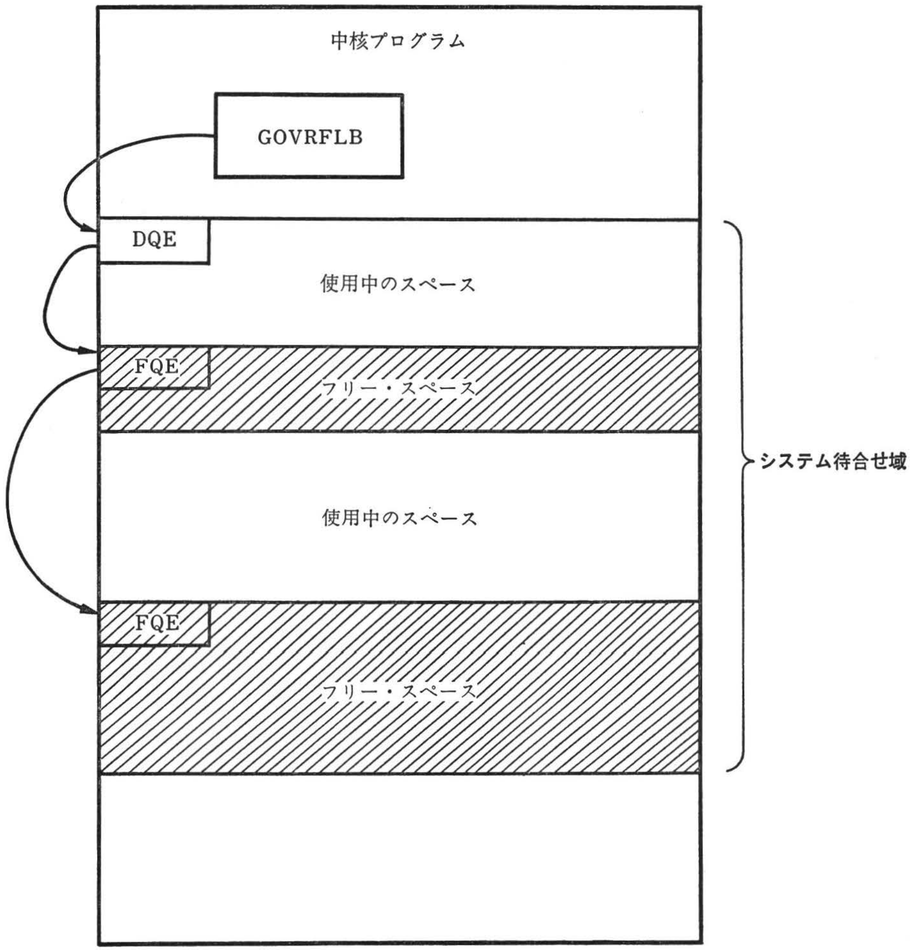
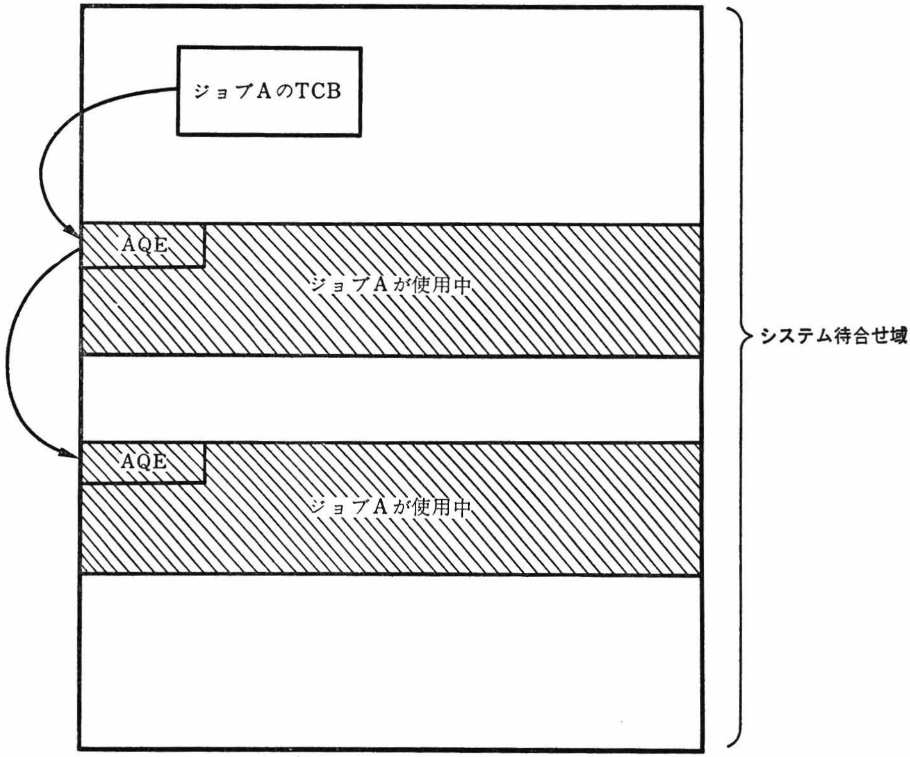


図 8-96　システム待合せ域のスペースの管理

ルは，つぎのような異った性質を持っているので，後述するようにその割り当てと解放の仕方も異る：

- サブプール 253：サブプール 253 のスペースは，明示的に解放要求がなされないかぎり，そのスペースを利用してるタスクが終了したときに自動的に解放される.
- サブプール 254：サブプール 254 のスペースは，明示的な解放要求がなさ



バイト		バイト	
0	(予約フィールド)	1	AQEPTR：つぎの割当て区域へのポインター
4	(予約フィールド)	5	長さ：当該割当て区域のバイト数

割当て待行列要素の形式

図 8-97　割当て待行列要素（AQE）によるスペースの管理

れないかぎり，そのスペースを利用しているジョブ・ステップが完了したときに自動的に解放される．

● サブプール 255：サブプール 255 のスペースは，必ず FREEMAIN マクロ命令を使って明示的に解放する必要がある．

ともあれ，システム待合せ域の空きのスペースの管理も，前述した領域内の空いたスペースの管理の場合と同じように，DQE と FQE を用いて行なう. DQE は，システム待合せ域の開始アドレスとその大きさ，および FQE チェーンへのポインターを持っている．一方，FQE はシステム待合せ域内の空きスペースに関する情報を持っている（**図 8-96** を参照).

オペレーティング・システムの生成時にシステム待合せ域として必要となるスペース量が指定される. NIP が実行されると，システム待合せ域に割り当てられたブロック数を収容した DQE がシステム待合せ域の下位アドレスに構築される．さらに，DQE に隣接して FQE が作られる．言うまでもなく，FQE には利用可能なスペース総量が表示されている（NIP が実行されている時点では，システ待合せ域内のすべてのスペースが利用可能状態にあるので，これを表示した FQE が作られるのである). なお，中核プログラムの中にある GOVRFLB には構築された DQE のアドレスを収容する.

一方，システム待合せ域内の使用中のスペースの管理は，**割当て待行列要素**（Allocated Queue Element；略して **AQE**）を通じて行なわれる．**図 8-97** で示しているように，AQE には使用中のスペースの大きさと後続 AQE へのポインターが収容されている．したがって，各タスクでシステム待合せ域から確保したスペースは AQE によって管理されるのであり，同一タスクのための AQE はそれぞれポインターで連鎖されているのである．また，最初の AQE は TCB からポイントされている.

1) システム待合せ域内のスペースの割当て

先述したように，システム待合せ域の要求はサブプール 253, 254, 255 に対して GETMAIN マクロ命令を出すことによって行なわれる．それぞれのサブプールによって，コントロールの仕方が異なるので，別々にスペースの割当て方法を説明してゆくことにする.

① サブプール 253 の割当て

GETMAIN マクロ命令でサブプール 253 が指示されていると，要求されたスペース・サイズに 8 バイトが付加され，利用可能なスペースの開始アドレス

が決定される．GETMAIN ルーチンは，要求区域の最初8バイトに AQE を作りあげ，要求バイト数に8バイトを加えた数を入れるのである．そのあとで，AQE を AQE の待行列に挿入する．換言するなら，AQE に所定のポインター・アドレスを挿入する．なお，AQE 待行列は，スペースを要求したタスクの TCB 内の TCBAQE フィールドでポイントされている．

該当タスクが終了すると，監視プログラムの終了ルーチンは，TCB 内の TCBAQE フィールドを起点に AQE 待行列を探索してゆき，そのタスクが確保したサブプール 253 のスペースすべてを解放するために FREEMAIN マクロ命令を出す．

② サブプール 254 の割当て

GETMAIN マクロ命令でサブプール 254 が指示されている場合にも，サブプール 253 の場合と同様の方法でAQE を作る．ただし，作りあげられた AQE は，スペースを要求したタスクの AQE 待行列ではなく，ジョブ・ステップの TCB からポイントされている AQE 待行列に挿入する．

これによって，ジョブ・ステップが完了したときは，ジョブ・ステップ用の TCB の TCBAQE フィールドを起点として AQE 待行列を探索し，そのジョブ・ステップで確保したサブプール 254 のスペースを解放するために FREEMAIN マクロ命令を出す．

③ サブプール 255 の割当て

GETMAIN マクロ命令でサブプール 255 を指定した場合には AQE は作らない．その代り，GETMAIN ルーチンは，要求ルーチンに対してレジスター1か，または指定された記憶域を使って割り当てた区域のアドレスを受け渡す．確保したスペースを FREEMAIN マクロ命令を使って解放する責任はすべて要求ルーチンにあるので，AQE は必要ないのである．

2) システム待合せ域内のスペースの解放

システム待合せ域内のスペースの解放も FREEMAIN ルーチンによって遂行される．FREEMAIN ルーチンは，システム待合せ域を示している DQE を最

初につきとめる．そのあとで，サブプール253と254のスペースの解放かどうかをチェックする．そうだとすると，解放すべきスペース・サイズに8バイトだけ加える（解放スペースにAQEも含めるためである）．一方，解放すべきスペースのアドレスから8バイトを引くことによって，AQEを含めたアドレスから解放するように調整する．

サブプール253と254のスペースを解放するさいには，該当AQEアドレスをTCBのTCBAQEフィールドから得る．このAQEで記述されている区域全体を解放するのであれば，該当AQEをAQE待行列から除去すだけでよい．AQEで規定しているスペースを部分的に解放するのだと，AQE内の長さフィールドを更新すればよい．

解放されたスペースが，システムに待合せ域内のすでに空いているスペースと接している場合には，合わせて1つの空いたスペースとして，それを反映するようにFQEを更新する．

6. ロールアウト／ロールインの処理機能

領域内のスペースの割当てに関する無条件要求がGETMAINルーチンで対処しきれないことが判明すると，GETMAINルーチンは**ロールアウト／ロールイン・モジュール**（rollout/rollin module）を連係する．この措置が可能となるのは，ロールアウト機能がシステムに組み込まれており，かつロールアウトを引き起こすことのできる資格がジョブ・ステップに与えられているときである．なお，スペースを要求したプログラムがシステム・ルーチンであるとか，スペース要求がシステム待合せ域のスペースに対するものであるとか，新規のジョブ・ステップを開始させるための領域を要求する場合には，ロールイン／ロールアウト・モジュールの連係は行なわなれない．

ロールアウト／ロールイン・モジュールは，要求を出しタスクおよびジョブ・ステップ内の他のタスクで使用するための一時的な追加領域を確保しようとする．要求ジョブ・ステップが現有の領域内で利用可能なスペース以上のものを要求した場合に対処するためには，このような処理が必要となる．ロールアウト／ロールイン・モジュールは，まず動的区域内の未割当てのスペースから

一時的領域を割り振ろうとする．不幸にして，十分な未割当てのスペースが存在しないとすると，別のジョブの中でロールアウト可能なジョブ・ステップを捜す．ジョブ・ステップの指名順位がスペースの要求を出したジョブ・ステップの指名順位よりも低く，TCB の所定フィールドでロールアウト可能標識がセットされており，かつ ENQ マクロ命令が出されたシステム資源を使用中であるとか待っているとかがない場合には，そのジョブ・ステップはロールアウトの対象となる．

ロールアウト／ロールイン・モジュールが目下の要求を満すに十分な大きさの領域を持ったロールアウト可能なジョブ・ステップをみつけると，実行中の入出力指令の完了を待ち，ペンディング中の入出力指令の実行を延期し，さらにペンディング中のオペレーター応答をも延期したうえで，選択されたジョブ・ステップ領域の内容をそのまま補助記憶装置に移してしまう（これがロールアウトである）．そのあとで，ロールアウトされた領域をスペース要求をしたジョブ・ステップに割り当てるために，所要の制御ブロックを作りあげ初期値化する．最後に，ロールアウト／ロールイン・モジュールはスペース要求プログラムに制御権を戻す．要求プログラムでは GETMAIN マクロ命令を再実行することによって，GETMAIN ルーチンを呼ぶ．GETMAIN ルーチンは，ロールアウトを通じて確保した領域から要求を満すためのサービスを行なう．

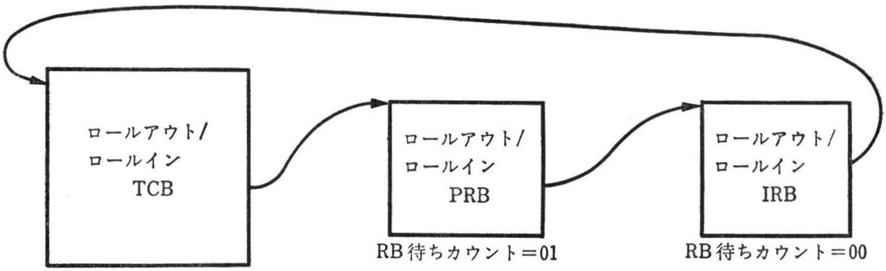
以上が，ロールアウト機能の概略であるが，以下でロールアウトの処理機能を詳しく説明してゆくことにする．

1）　ロールアウト／ロールイン・タスクのスケジューリング

ロールアウト／ロールインの機能は，NIP が attach したロールアウト／ロールイン・タスクによって遂行される．このタスクは常駐システム・タスクであり，非常に高い指名順位が与えられている．

ロールアウト／ロールイン・タスクには PRB と IRB とがそれぞれ1個ずつチェインされており，PRB は常に待ちカウントが1セットされており，待ち状態にある，一方，IRB の待ちカウントは0にセットされており，作動可能状態であるが，TCB，PRB をポイントしているためタスクそのものは待ち状

① ロールアウト/ロールイン・タスクがディスパッチ不能状態



② ロールアウト/ロールイン・タスクがディスパッチ可能状態

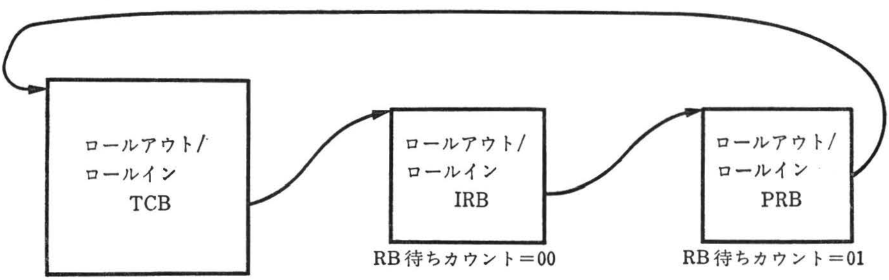


図 8-98 ロールアウト/ロールイン・タスクのスケジューリング中の TCB, PRB, IRB の相互関係

態におかれている(図 8-98 の①を参照). ところが, ロールアウトないしはロールインの必要性が生じると, PRB と IRB の順序を逆にすることによって, ロールアウト/ロールイン・タスクが作動可能な状態になる(図 8-98 の ② を参照).

ロールアウト/ロールイン・タスクのスケジュールは, 非同期的エグジット・ルーチンで行なわれる. 非同期エグジットの仕組みに関しては,「四. タスク監視プログラムの機能構造」で詳述しているので必要に応じて参照されたい. ロールアウト/ロールイン・タスクをスケジュールする場合の制御権の流れを示したのが図 8-99 である. この図でも分るように, ロールアウト/ロールイン・タスクのスケジュールは大きく 2 つのステップに分けて考えることができ

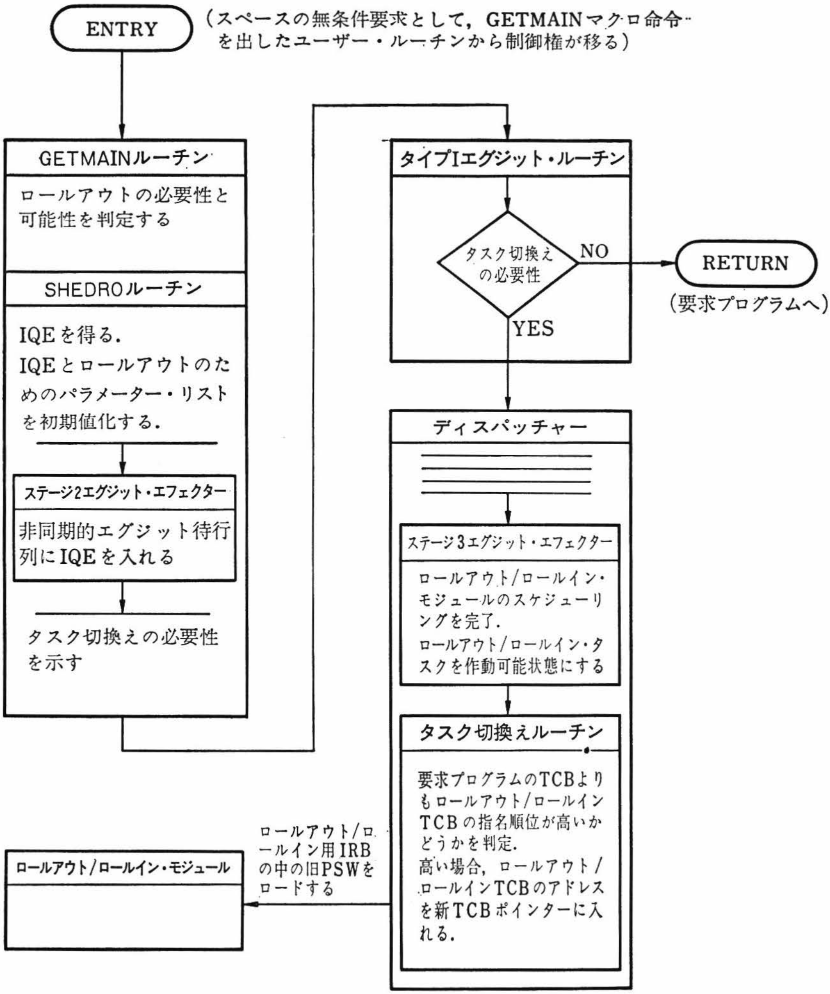


図 8-99　ロールアウト／ロールイン・タスクのスケジューリングの流れ

る．つまり，GETMAIN ルーチンのサブルーチンである SHEDRO ルーチンによる初期的スケジューリングとステージ3エグジット・エフェクターによる最終的スケジューリングとである．

① SHEDRO ルーチンによる初期的スケジューリング

ロールアウト／ロールインを行なう場合，GETMAIN ルーチンは SHEDRO ルーチンをサブルーチンとして使用する．SHEDRO ルーチンは，IQE およびロールアウトのためのパラメータ・リストを獲得して，それぞれの各フィールドを初期値化する．IQE とパラメータ・リストの記憶域は NIP 時にとられ，IQE はロールアウト／ロールイン用の IRB からポイントされる．

つぎに，ステージ 2 エグジット・エフェクターを使って，ロールアウト／ロールイン・タスクに関する IQE を非同期エグジット待行列に入れる．ロールアウト／ロールイン・タスクを活動状態にするためには，ステージ 3 エグジット・エフェクターを働かす必要があり，その為にはタスクの切換えを引き起す必要がある．なお，ステージ 3 エグジット・エフェクターはディスパッチャーのサブルーチンとして働く．

また，ロールアウトが完了するまで GETMAIN マクロ命令を出したタスクをディスパッチ不能状態におき，ロールアウトが完了したのちタスクを再開始したとき再び GETMAIN マクロ命令を出すようにする．このため，つぎの 3 点の処理を行なう：

- IEATCBP の新 TCB ポインターを 0 にして，タイプ I エグジット・ルーチンからディスパッチャーに制御権が渡るようにする．これによって，GETMAIN ルーチンからタイプ I エグジット・ルーチンを経てディスパッチャーに制御権が移る．
- GETMAIN マクロ命令を出したタスクをディスパッチ不能状態におく．
- SVC 旧 PSW の中にある“つぎの命令アドレス(next instruction address)”フィールドから 2 だけマイナスし，これによってタスクが再スタートしたとき，再び GETMAIN マクロ命令を出すようにする．

② ステージ 3 エグジット・エフェクターによる最終的スケジューリング

GETMAIN ルーチンからタイプ I エグジット・ルーチンに制御権が移ると，エグジット・ルーチンでは IEATCBP の新 TCB ポインターが 0 であることからタスク切換えの必要性を知り，ディスパッチャーへ制御権を渡す．ディスパッチャーは，非同期的エグジット待行列に IQE があることを知り，非同期的

エグジット・ルーチンをスケジュールするために，ステージ3エグジット・エフェクターへ行く．この場合の非同期的エグジット・ルーチンとは，ロールアウト／ロールイン・モジュールのことである．そこで，ステージ3エグジット・エフェクターは，ロールアウト／ロールイン・モジュールをスケジュールするためにつぎのことを行なう：

- ロールアウト／ロールイン用 IQE を非同期的エグジット待行列から取除き，ロールアウト／ロールイン用 IRB からポイントされる活動状態の待行列に加える．IQE は，ロールアウト／ロールイン・モジュールが行なうべきことの情報を収容した制御ブロックであり，ロールアウト／ロールイン・モジュールの実行が終るまで IRB からポイントされている．ロールアウト／ロールインの実行が完了すると，IQE は活動状態の待行列から除去され，同じく IRB からポイントされている別の待行列に移される．
- ロールアウト／ロールイン用 TCB の PRB と IRB のチェイニング順序を逆にして，ロールアウト／ロールイン・タスクを作動可能状態にする．
- ステージ3エグジット・エフェクターは，タスク切換えルーチンを使って IEATCBP の新 TCB ポインターにロールアウト／ロールイン用 TCB のアドレスを入れる．これによって，ディスパッチャーはつぎにロールアウト／ロールイン・タスクをスタートさせることができるようになる．
- IQE にあるパラメータ・リストのアドレスをレジスター1に入れ，その値がプラスか，マイナスかあるいはゼロかによってロールアウト／ロールイン・モジュールに実行すべき機能を知らせる．レジスターの値がプラスだとロールアウトを，マイナスだとロールインを，さらにゼロだと延期されているロールアウトを再スケジュールすることを意味する．

なお，ロールアウト／ロールイン用タスクのスケジューリングの様子はすでに図 8-51 に示しているので参照して欲しい．

2) ロールアウトの処理構造

ロールアウトとは，現有領域で使用できるスペース以上のスペースを必要と

するジョブ・ステップに追加の領域を一時的に割り当てようとする試みである．ロールアウト／ロールイン・モジュールは，まず動的区域内の空きのスペースから追加の領域を割り当てようとする．動的区域に十分な空きスペースがないと，別のジョブ・ステップの領域の内容をそのまま補助記憶装置に書き出し，その領域を借りる．そのあとで，借りた領域をスペースを要求したジョブ・ステップに割り当てる．

ロールアウトにさいして，ロールアウト／ロールイン・モジュールが行なう主要な機能は，つぎの7種類である：

① ロールアウトを行なうべきかどうかの判定を行なう．
② 動的区域の未割当ての記憶域から必要なスペースを確保する．
③ ロールアウトするにふさわしいジョブ・ステップと領域とをみつけ出す．
④ ロールアウトにふさわしいジョブ・ステップがみつかった場合の処理．これには，選択された領域の内容がすでにロールアウトされているが，その領域が未使用の場合の割当て処理も含まれる．領域の内容がまだロールアウトされていない場合だと，ロールアウトするジョブ・ステップのタスクをディスパッチ不能状態にし，入出力要求を延期し，かつオペレーターの応答をも延期するといった処理が含まれる．
⑤ 選択された領域の内容をロールアウト・データ・セットに移す．
⑥ 借りた領域をスペースを要求したタスクのジョブ・ステップに割り当てる．
⑦ ロールアウト／ロールイン・モジュールからのエグジットのための諸準備をする．

これら，ロールアウトにさいしてのロールアウト／ロールイン・モジュールの諸機能に関して以下で順を追って説明してゆくことにする．

① ロールアウトの可否判定

GETMAIN マクロ命令が出され，その要求を満すだけ十分なスペースがない

場合，マクロ命令を出したタスクにロールアウトを行なう資格があればロールアウト／ロールイン・タスクが活動状態にされる．ロールアウト／ロールイン・タスクのもとで働くロールアウト／ロールイン・モジュールは，レジスター1で渡されたパラメータ・リストのアドレスを調べ，その値が正であればロールアウトが要求されたと判断する．

ロールアウトが要求されていると，**“ロールアウト呼び出し**(rollout invoked)”カウンターを調べて，すでに別のジョブでロールアウトが引起され，それが生きたままの状態であるかどうか確める．インターロック状態におちいることを避けるため，一時点で2つ以上のジョブ・ステップが同時にロールアウトを行なうことを禁じているからだ．ただし，ユーザーが作成したロールアウトの付加ルーチンで**多重ロールアウト**（multiple rollout）を許している場合だと，他のジョブ・ステップがすでにロールアウトを行なっている状態であっても，さらにロールアウトをすることができる．

ともあれ，多重ロールアウトが禁じられているシステムで，別のジョブ・ステップがロールアウトを引起し，いまだそれが有効であると，目下のロールアウト要求は延期される．延期された要求は，ロールアウトが可能になった時点で再スケジュールされる．別のジョブ・ステップでロールアウトを行なっていないとか，ロールアウトが行なわれていてもユーザーの付加ルーチンで多重ロールアウトを許している場合だと，目下のロールアウト要求の処理がつづけられる．

②　動的区域内の未割当て記憶域からのスペースの確保

ロールアウトが可能であることがわかると，まず動的区域内の未割当てのスペースが十分あるかどうか調べる．このために，サブプール248を指定した条件つきの GETMAIN マクロ命令を使って，動的区域の未割当てのスペースから新規の領域を得ようとする．十分なスペースがなくて戻りコード4が返ったときは，ロールアウトすべきジョブ・ステップを捜す作業に入る．十分なスペースがみつかって，一時的な領域が割り当てられると戻りコード0が戻される．この場合には，GETMAIN ルーチンでスペースを割り当てるだけでなく，

それに対応する PQE と FBQE を作りあげ，PQE をロールアウト／ロールイン用の TCB の待行列に入れる．

動的区域の空きのスペースから新しい領域を確保したさい，ロールアウト／ロールイン・モジュールが行なう具体的処理はつぎのものに集約できる：

- 新しく確保した領域を示すための PQE を，スペース要求を出したタスクの TCB でポイントしている PQE 待行列に入れる．また，PQE の中の TCB アドレス・フィールドを 0 にして，空いたスペースから領域をとったことを示す．換言するなら，ロールアウトを行なわなかったことを示す．これによって，ロールインのさいには，領域を解放するだけで十分であることがわかる．さらに，PQE に **“借りた** (borrowed)” **旨**の標識を立てる．
- “ロールアウト呼び出し” カウンターに 1 を加える．
- 新しい領域を得たジョブ・ステップ・タスクの TCB に “ロールアウト呼び出し” 標識を立てる．
- ロールアウト／ロールイン・モジュールからエグジットする．

③ ロールアウトすべきジョブの選択

動的区域内の空きのスペースから新規の領域を割り当てることができないと，ロールアウト／ロールイン・モジュールは，ロールアウトするにふさわしいジョブ・ステップを得ようとする．つぎの条件を満足した場合，そのジョブ・ステップはロールアウトの対象となる：

- 該当ジョブ・ステップが他のジョブをロールアウトしていない．
- 該当ジョブ・ステップの TCB にロールアウトできる旨の標識が立てられている．
- 該当ジョブ・ステップの領域が，要求スペースより大きい．

これらの条件を満たすジョブ・ステップを指名順位の一番低いジョブ・ステップから順番に捜す．ただし，ロールアウトを要求したジョブ・ステップより高い指名順位を持ったジョブ・ステップに関しては，ユーザーの付加ルーチン

で許可していないかぎり，ロールアウトすることはできない．なお，ロールアウトすべきジョブ・ステップがみつからない場合の処理はつぎの通りである：

- **異常終止付加ルーチン**（ABEND appendage）が，スペース要求を出したジョブ・ステップ・タスクを異常終止させる．
- スペースの要求を延期し，異常終止付加ルーチンが他のジョブ・ステップを選択し，それを異常終止させる．
- 異常終止付加ルーチンが組み込まれていないと，スペース要求を延期する．

④　ロールアウトするための準備

ジョブ・ステップをロールアウトするための前準備として，つぎの3つのことを行なう：

- ジョブ・ステップ内のすべてのタスクをディスパッチ不能状態にする．これは，それぞれのタスクのTCB内にあるTCBFRO標識を立てることによって達成される．
- 対象ジョブ・ステップが出した入出力要求を延期する．IOSの待行列に入れられている入出力要求が，ロールアウト後に開始されるとそのジョブに属さない区域にデータを読み込んだり，逆にジョブに属さない区域からデータを書き出したりする可能性がある．このために，IOSの待行列に置かれている入出力要求をPURGEマクロ命令を使って延期するのである．延期された入出力要求に対応するIOBの様子を示したのが**図8-100**である．延期された入出力要求は，あとでロールインが行なわれたときにリストアされる．なお，すでに開始されている入出力オペレーションに関しは，その終了を確認したあとでロールアウトを遂行するようにしている．
- 対象ジョブ・ステップに対するオペレーターの応答を延期する．ジョブ・ステップがロールアウトされている最中に，オペレーターがロールアウトされたジョブに返した応答は領域内のバッファーに入れることはできない．そこで，ロールインしたあとでその応答を領域内のバッファーに入れ

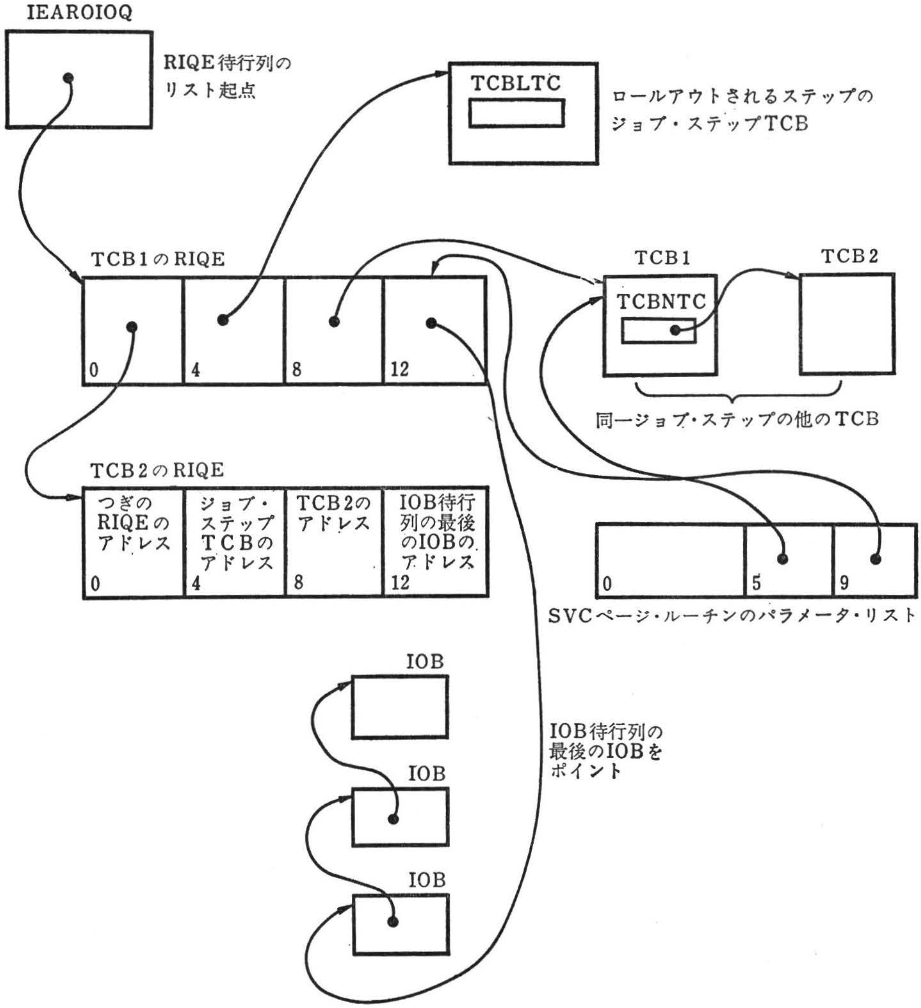


図 8-100 延期されたI/O要求に対応するIOBの待行列への入れ方

るようにするための前準備として，応答内容を一時的バッファーに保管しておく．オペレーター応答がこの様に処理されるための前準備をするために，**応答延期ルーチン**（reply purge routine）にブランチする．なお，応答内容の保管の面倒はコミュニケーション・タスクがみる．

⑤ ロールアウト・データ・セットへのロールアウト

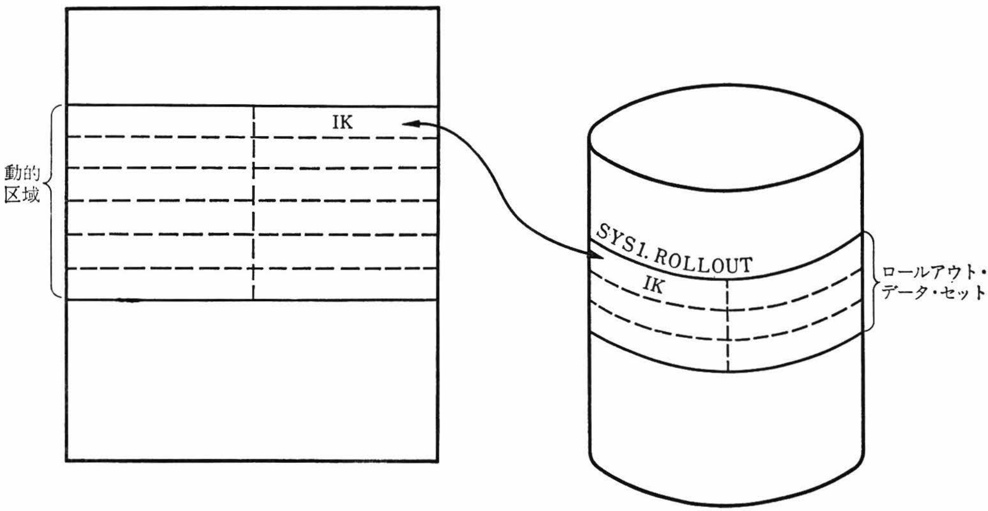


図 8-101　ロールアウト・データ・セットへのロールアウト

領域内の内容のロールアウトは，ロールアウト／ロールイン・モジュールのうちの start transfer ルーチンが担当する．ロールアウトする領域に対応する PQE のアドレスが，パラメータとして start transfer ルーチンに渡される．PQE がわかれば，ロールアウトする領域の開始アドレスとその大きさがつかめるのである．

ロールアウト・データ・セット（SYS 1. ROLLOUT）はあらかじめカタログされている必要がある．このデータ・セットは，IPL のさいに1Kバイトのブロック・サイズで初期値化される．したがって，動的区域内のスペースとロールアウト・データ・セットのレコードとは，1Kバイト単位で1対1の対応づけがなされる（図 8-101 を参照）．PQE で示されている領域の開始アドレスから始まり，領域の最終アドレスまでの内容が1Kバイト単位でロールアウト・データ・セット上の対応するレコードに書き出されるのである．領域内の1Kバイトのスペースとロールアウト・データ・セット上の対応は，つぎの式によって関係づけられる：

$$TTR=((R_0-K_1)/R_1)/N$$

R_0：ロールアウトする1Kバイト・ブロックの領域内のアドレス．

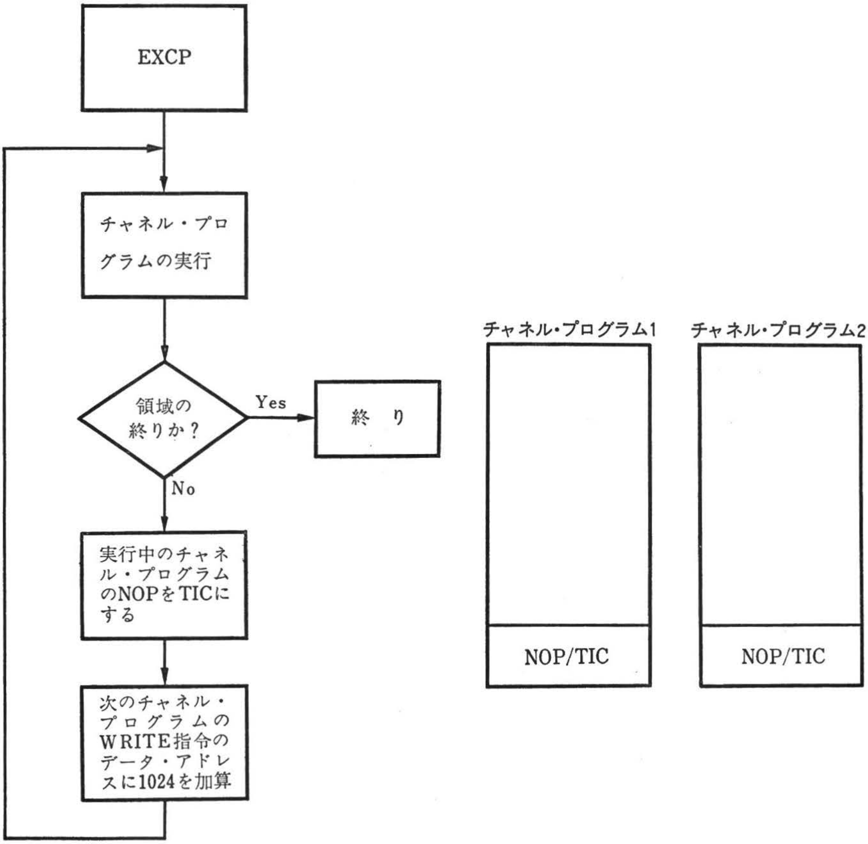


図 8-102　ロールアウトの制御権の流れ

K_1：動的区域の最下位アドレス．このアドレスはGOVERFLBに収容されている．

R_1：レコード・サイズ（1Kバイト）．

N；ロールアウト・データ・セット上の1トラックに収容できる1Kバイト・レコードの個数．

実際のロールアウトは，2つのチャネル・プログラムを使用してPCIにより交互にチェーニングしながら行なわれる．この場合の制御権の流れを示したのが**図 8-102**である．この図で示されているように，1回の EXCP でロール

アウトが完了する．PCIによるチャネル・プログラムの接続がうまくいかなかった場合，またはシリンダーの終りなどで**コマンド（指令）連鎖**（command chaining）が切れたときは，付加ルーチンからIOSのEXCPの再試行を行なうアドレスに戻り，チャネル・プログラムを再開始させる．付加ルーチンとしては，チャネル終了付加ルーチン，異常終止付加ルーチンを使用する．

EXCP命令を出したあと，ロールアウト／ロールイン・タスクは待ち状態に入る．領域の内容が完全にロールアウト・データ・セットに書き出されるとロールアウトECBがPOSTされ，ロールアウト／ロールイン・タスクは作動可能状態に戻される．

⑥ ロールアウトで獲得した領域を要求タスクに割当てる方法

つぎの方法で，ロールアウトされたことによって得られた領域を割り当てる：

- 領域の元の所有タスクのPQE中に“ロールアウト”標識と“使用中”標識とを立てる．
- 新規のPQEを作りあげ，そのPQE内のFBQEへのポインターがロールアウトされた領域のFBQEをさすようにする．一方，領域内のTCBへのポインターは，その領域の本来の持ち主のTCBをポイントしている．
- ロールアウトされた領域の記憶保護キーをゼロにセットする．
- “ロールアウト呼び出し”カウンターに1を加える．PQEに“使用した”旨の標識を立て，新しい使用タスクのTCBからポイントされているPQE待行列に加える．さらに，新しい使用タスクのジョブ・ステップ・タスクのTCBに“ロールアウト呼び出し”標識を立て，ディスパッチ不能標識をリセットする．

このような処理を行なったあと，ロールアウト／ロールイン・モジュールからエグジットするために，RETEXITルーチンへブランチする．

⑦ ロールアウト／ロールイン・モジュールからのエグジット

ロールアウト／ロールイン・モジュールからエグジットするために，まず，IQEを活動状態の待行列から取はずし，別の待行列に移す．ロールアウトの要

求が延期された場合だと，すでに IQE は活動状態の待行列から取除かれているので，この処理は不要である．

このあとで，IEATCBP の新 TCB ポインターを0して，タスク切換えが生じるようにする．タスク切換えのセットをしたあと，SVC 3 を通じて監視プログラム・エグジット・ルーチンにブランチする．この結果，ロールアウト／ロールイン・タスクの TCB がポイントしている IRB と PRB のチェーングの順序が入れかわり，ロールアウト／ロールイン・タスクの TCB はディスッチ不能状態に戻されるのである．

3) ロールインの処理構造

ロールアウトの要求を出したジョブ・ステップが借用領域の使用を完了すると(これは FREEMAIN マクロ命令が実行されることによって知らされるが)，FREEMAIN ルーチンがロールアウト／ロールイン・モジュールを連係する．ロールアウト／ロールイン・モジュールは，ロールアウトした内容を補助記憶装置から主記憶装置内のもともと割り振られていた場所に戻す．これがロールインである．そのあとで，ロールインのさいに延期しておいた入出力指令とオペレータ応答とをそのジョブ・ステップにリストアする．これらの処理が終ると，ロールアウト／ロールイン・モジュールは，エグジット・ルーチンとディスパッチャを経由して，最も指名順位の高い作動可能状態のタスクに制御権を戻す．

ロールアウト／ロールイン・モジュールは，入力パラメータ・リスト（つまり，レジスタ 1）がマイナスであることによってロールインの必要性を判定する．その結果，ロールアウト／ロールイン・ルーチンにブランチする．このルーチンは，ロールインにさいして遂行するすべての機能の調整を行なう．ロールインのさいに行なわれる機能としてはつぎのものがある：

- 領域を使用したジョブ・ステップの PQE で占められていたスペースを解放する．
- ロールアウトされている ジョブ・ステップの ロールインをすべきかどう

か判定する．このために，該当ジョブ・ステップの PQE を調べる．PQE の“借りた (borrowed)”旨の標識がオンだと，その領域はロールアウトによって借りたものであることを示している．PQE の“借りた”旨の標識と，“解放ずみ”の標識の双方がオンだと（つまり，借用領域でしかもその使用が終了した時），その領域は，目下の使用タスクから解放し，ロールインを行なう．なお，領域の解放は，目下の使用タスクの PQE 待行列の中から，解放する領域用の PQE をはずすことによって達成される．

- ロールインすべきことが判定されると，ロールアウトしていたジョブ・ステップを主記憶装置内の元の場所に戻す．この場合，記憶保護キーを元のジョブ・ステップのそれに戻す．
- ロールインしたジョブ・ステップのために，延期しておいた出入力要求を再開させる．
- 同様に，延期しておいたオペレータ応答も再開させる．
- ロールインしたジョブ・ステップ用のタスクをディスパッチ可能状態にする．
- さらに，延期されているロールアウト要求があれば，そのロールアウトをスケジュールする．

なお，動的区域の空きのスペースから該当領域を借りている場合だとロールインは行なわない．領域の本来の持ち主の PQE から“使用中”標識をリセットするだけである．

以下でロールインに関する主要な機能に関して補足説明をしておこう：

① ロールアウト・データ・セットからのロールイン

ロールインは，ロールアウト／ロールイン・タスクによって遂行されるが，ロールアウト／ロールイン・タスクをスケジュールする方法は，前述したロールアウトのさいのやり方と同様である．また，ロールアウト・データ・セットからロールインする時の入出力オペレーションもロールアウトする場合と同様である．ただし，入出力指令のコードが READ になる点だけが異なる．実際のロールインは，下記の手順で行なわれる：

- 該当領域の記憶保護キーをロールインされるジョブ・ステップ・タスクのキーにセットする.
- ジョブ・ステップをロールアウト・データ・セットからロールインする.
- そのジョブ・ステップ・タスクのPQEの“ロールアウト”標識をリセットする.
- 領域用の空きのスペースの記憶保護キーを0にセットする.
- ロールインの最中に入出力エラーが生じた場合には, ABTERMルーチンを使ってロールインを試みたタスクの異常終止をスケジュールする.

② 入出力要求のリストア

ロールアウトされている間延期されていた入出力要求は, ロールインされたときにリストアされる. 入出力要求のリストアはRESTOREマクロ命令によって遂行される. このマクロ命令が実行されると, 延期されていた入出力要求はマクロ命令を出したタスクに連結される. つまり, RESTOREマクロ命令を出したタスクが入出力要求を出したのと同じ結果になる. したがって, ロールアウト／ロールイン・スタクではなく, ロールインされたタスクがRESTOREマクロ命令を出した形にする必要が生じる. これは, **図8-103**で示しているように, ロールアウト／ロールイン・タスクのIRBをロールインされたタスクのTCBに連鎖することによって行なわれる.

ロールインされたタスクは, IRBが連鎖されている間だけTCBの“ディスパッチ不能”標識がリセットされ, 一時的に作動可能状態となる. 一方, ロールアウト／ロールイン・タスクは, IRBがTCBチェーンからはずされている間は待ち状態にあるPRBを指しているために必然的に待ち状態に置かれることになる. なお, ロールアウト／ロールイン・タスクとロールインされたタスクとの間のタスク切換えは, IEATCBPの“新PSW”を0にしてディスパッチャにブランチすることによって行なわれる.

入出力要求のリストアは, ロールインされたジョブ・ステップ内のすべてのタスクについて実施される. ロールインされたジョブ・ステップ内のタスクへのポインターは図8-100に示した通りである.

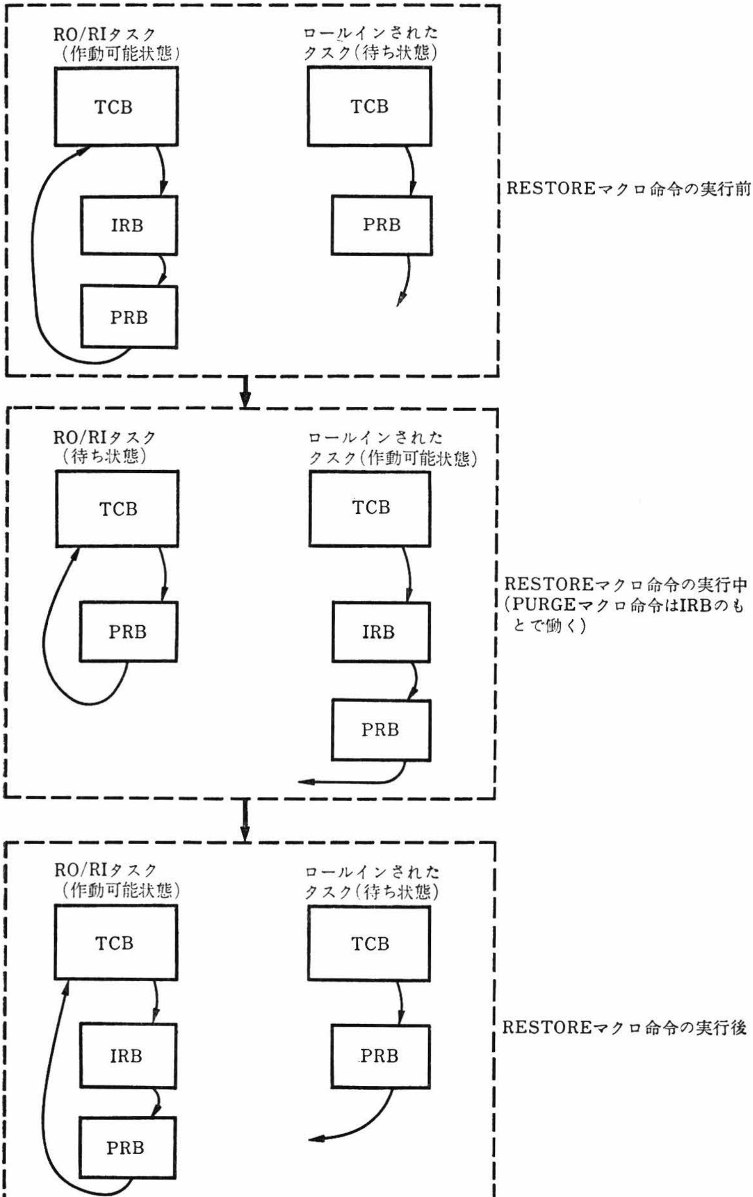


図 8-103　入出力要求のリストア

③ 延期されたロールアウトの再スケジュール

すでにシステム内でロールアウトが行なわれている場合とか，十分な大きさを待った領域がない場合には，ロールアウトの要求は延期されてしまう．これについては，「ロールアウトの処理構造」の個所でふれた通りである．

ロールインが終了すると，延期されているロールアウトの再スケジュールが行なわれる．ロールアウト要求待行列に IQE があると，各々の IQE につぎの処理を加えることによって再スケジュールの準備をする：

- ロールアウトを要求しているタスクの“ディスパッチ不能”標識をリセットする．
- ステージ２エグジット・エフェクターが IQE を非同期エグジット待行列に入れる．

ロールアウトのスケジュールの仕方は，「ロールアウト／ロールインのスケジューリング」の個所で説明した通りである．なお，ロールアウト要求待行列の全ての IQE を処理すると，ロールアウト／ロールイン・モジュールからエグジットする．

これまで説明してきたように，主記憶装置監視プログラムは大別して４種類の記憶域の割当てと解放の機能を遂行してくれる．つまり，領域の割当てと解放，領域内のスペースの割当てと解放，システム待合せ域内のスペースの割当てと解放，さらにはロールアウト／ロールインに関する領域の割当てと解放の機能である．これらの機能を遂行するために，PQE, FBQE, SPQE, DQE, FQE あるいは AQE などの各種制御ブロックを利用し，保守してくれるのである．

七．タイマー監視プログラムの機能構造

1. タイマーの管理

タイマーの管理の仕方は，ハードウェア・システムに賦与されているタイマー機能の差によって若干異なる．したがって，ここでは IBM システム/360とシステム/370とをとりあげ，それぞれのタイマーの管理について概観してゆくことにする．

1）システム/360のタイマー監視プログラム

システム/360では，**間隔計時機構**（interval timer）を用いてタイマーに関するサービスを行なう．システム/360の間隔計時機構は32ビット（1ワード）の大きさで主記憶装置に確保されており，システムが作動中でかつ間隔計時機構のスイッチがオンであると自動的に1ずつ減少する仕組みになっている．この間隔計時機構の機能を用いて，各種のサービスを果すのがタイマー監視プログラムである．**タイマー監視ルーチン**（timer supervision routine）は，プログラマーが**日時**（date and time of day）を得るとか，時間間隔を計るとか，あるいは特定時間の活動をスケジュールするとかのためのマクロ命令に対してサービスを提供する．

タイマー・サービスの要求は，割込みを通じて知らされる．その結果，自動的に所要のタイマー監視ルーチンに制御権が渡される．TIME，STIMER，TTIMER マクロ命令が実行されると SVC 割込みが生じる．一方，時間計時機構が正から負に変るとタイマー割込みが生じる仕組みになっている．なお，SVC 割込みが生じると，3種類のマクロ命令のサービス・ルーチンのうちのいずれかに制御権が移ることになる．それぞれのサービス・ルーチンについては後で詳述するが，各々の機能の概略について以下でふれておく：

① **TIME ルーチン**：現時点の日付と時刻とを提供してくれる．最初はオペレータが SET 指令を用いて開始の日付と時刻を与える。そのあとは，タイマー監視ルーチンが，真夜中を過ぎたときに日付を変更し，経過時刻を管理してくれる．一方，TIME ルーチンは現時点の日付を入手し，オペレータがセットした開始時刻に経過時間を加算したうえで，日付と時刻とを汎用レジスターに戻してくれる．

② **STIMER ルーチン**：**時間間隔**（timed interval）の要求を処理する．つまり，間隔計時機構にその値を挿入して，要求時刻に割込みが生じるようにスケジュールする．STIMER マクロ命令が出されるつど，**待行列要素**（queue element）を作りあげ，STIMER マクロ命令の中の情報を集約して入れてくれる．そのあとで，経過時間を基準にして，待行列要素をタイマー待行列に

配置する．所定の時間が経過してタイマー割込みが生じると，**タイマー割込み処理ルーチン**（timer interruption handling routine）が要求したアクションの処理を行ない，タイマー待行列の要素の中から新しい時間間隔を得て間隔計時機構にセットする．

③ **TTIMER ルーチン**： あらかじめ STIMER マクロ命令で設定された時間間隔の残り時間をユーザーに返したり，残り時間に対応する先行要求を取消す．残り時間を決定するために，TTIMER ルーチンでは，時間間隔の完了時間から経過時間を引く．先行要求を取消すためには，タイマー待行列から対応する待行列要素を除去する．

2） システム/370のタイマー監視プログラム

システム/370 では，間隔計時機構と **TOD**（Time-Of-Day）**クロック**の２つのタイマーを用いて，タイマー管理を行なっている．

TOD クロックは，システムに電源が入っている間は継続して働くのでIPLのつどセットする必要はない．TODクロックは，自動的に経過時間を管理してくれる（1960年１月１日の０時をベースにして）ので，TIME ルーチンではTODクロックを用いて現時点の日付を更新し，時刻を決定する．現時点の日付と日時とは，TIME マクロ命令で指示した方法で呼び出しプログラムに渡される．

要求時間間隔が１時間より大きい場合だと，STIMER ルーチンも TOD クロックを使用する．要求間隔が１時間以内だとシステム/360と同じ処理がなされる．

時間間隔の残り時間を決定するために，TTIMER ルーチンは完了時に TOD クロックで示されると想定される値から現時点の TOD クロックの値を引く方法をとっている．

2. タイマー SVC 割込みの処理手順

タイマーに関するマクロ命令が出された時の SVC 割込みの処理は，図 8-104 で示している流れにもとづいて遂行される．

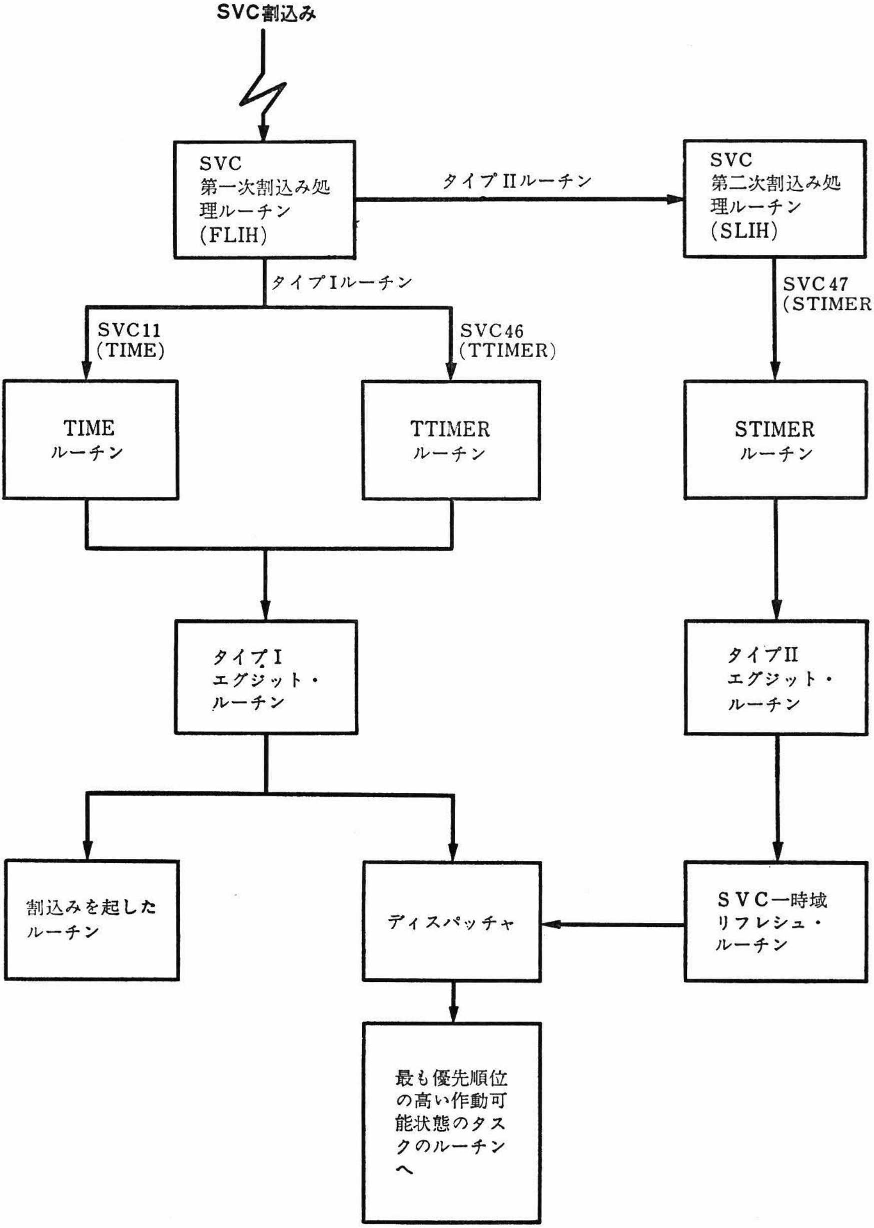


図 8-104　タイマー SVC 割込み処理

タイマー SVC 割込みが生じると，SVC FLIH が制御権を得て割込みを引起したプログラムに関する情報を貯える．そのあとで，所要ルーチンに制御権を渡す．TIME と TTIMER マクロ命令を実行することによって生じた SVC 割込みの場合には，直接サービス・ルーチンに行く．一方，STIMER マクロ命令の場合は，SVC SLIH に制御権が渡る．SVC SLIH は，監視プログラム要求ブロック（SVRB）を作り，そこに必要な情報を貯えたあとで，STIMER ルーチンに制御権を移す．

タイプⅠ SVC ルーチンである TIMER ルーチンおよび TTIMER ルーチンの実行が終ると，タイプⅠエグジット・ルーチンに制御権が渡り，以後の処理を決定する．割込みを起したルーチンに戻る場合には，前に貯えてあったレジスターの中味を復元し，該当ルーチンに制御権を戻す．他のタスクに制御権を渡す場合は，レジスターの中味を割込みを起したプログラムの TCB に貯え，ディスパッチャに制御権を移す．

タイプⅡの SVC ルーチンである STIMER ルーチンの実行が終ると，エグジット・ルーチンを経由してディスパッチャに制御権が移る．エグジット・ルーチンでは，タイプⅠエグジット・ルーチンとほぼ同じ処理を行なう．ディスパッチャは，その時点で最も優先順位の高い作動可能状態のタスクに制御権を渡す．

3. TIME ルーチンの機能構造

ある時点の日付と時刻の情報を要求するのに用いるが，TIME マクロ命令で

表 8-27 TIME マクロ命令の一般形式

〔記　号〕	TIME	{ DEC（下線） / BIN / TU / MIC, アドレス }

- DEC： HHMMSSth（t は10分の1秒，h は百分の1秒）の形式でレジスター 0 に時刻が戻される
- BIN： 符号なしの32ビットの2進数形式でレジスター 0 に時刻が戻される（100分の1秒の単位で）
- TU： BINと同じ，ただし，26.04166マイクロ秒（タイマーの単位時間）の単位で表示される．
- MIC, アドレス： アドレスで指定された8バイトの記憶域に符号なしの2進数形式で時刻が戻される（ただし，51ビット目がマイクロ秒に等しい形式で）

ある．このマクロ命令が実行されると，TIME SVC ルーチンでは日付をレジスタ1に，時刻をレジスタ0に入れてくれる．TIME マクロ命令の一般形式は，**表 8-27** に示した通りである．

レジスタ1に戻される日付は，パック10進数で 00 yydddc の形式である．yy は西暦年号の下2桁であり，ddd はその年の累積日数である．c は16進数Fで示される符号である．この符号があることによって，日付情報を直接アンパックすることができ，印刷などの出力に使える．なお，日付を要求する簡単な例を示したのが**図 8-105** である．

レジスタ0に戻される時刻の形式は，TIME マクロ命令のオペランドの指示にしたがう．TIME マクロ命令のオペランドで BIN を指定すると，100分の1秒を単位とした経過時間を示す数字が符号なしの32ビットの2進形式で戻される．一方，TU と指定するとタイマーの時間単位（つまり，26.04166 マイクロ秒）にもとづいた経過時間を示す数字が符号なしの32ビットの2進形式で戻される．また，DEC と指定するか，オペランドを省略してしまうと，パック10進数で HHMMSSth（H：時間，M：分，S：秒，t：10分の1秒，h：百分の1秒）の形式の情報が戻される．

以上のように，BIN，TU，DECオペランドを指定した TIME マクロ命令を与えると，日付はレジスタ1に，時刻はレジスタ0に戻される．しかしながら，オペランドでMIC，アドレスの指定をすると時刻は“アドレス”で指示した記憶域に符号なしの64ビットの2進数形式で戻される．この場合の時刻は51ビット目がちょうど1マイクロ秒に等しい値となっている．日付は他の指定と同じくレジスタ1で示される．なお，レジスタ0はゼロにセットされてしまう．

```
           :
         TIME                     日付の要求
         ST     1, ANS            パック10進数の日付けを ANS に格納
         UNPK   DOUBLE, ANS       印刷するために日付けをゾーン10進数に
           :
ANS      DS     F                 パック10進数の日付け用のフルワード
DOUBLE   DS     D                 ゾーン10進数の日付け用のダブルワード
```

図 8-105　日付けの処理例

以下で TIME マクロ命令の実行によって機能する TIME ルーチンの機能構造をみてゆこう．

1）システム/360の TIME ルーチン

TIME ルーチンは，いうまでもなくその時点の日付と時刻を入手し，要求ルーチンに両者の値を戻す機能を果たす．日付は，**CVT**（communication vector table）の中の CVTDATE フィールドから取り出し，レジスター1に入れる．このフィールドに日付を入れるのは，SET 指令が出された時のジョブ管理プログラムの中の SET 指令ルーチンの機能である．なお，真夜中が過ぎたときの日付の更新は，**タイマー第二次割込み処理ルーチン**（timer second-level interruption handler）が行なう．一方，時刻の決定にさいしては，オペレータが SET 指令で設定した値に経過時間を加算することによって計算を行なう．

開始時刻がセットされたあとは，時間計時機構が絶えず作動しているので，これをもとに経過時間を把握することができる．時間計時機構は自動的にその値を1ずつ減じてゆき，その値が負に変ったときに割込みを引起す．そこで，時間保存の目的のために6時間間隔を採用している．IPL 時に時間計時機構に6時間に相当する値が入れられる．この値が負になると，別の6時間間隔がセットされる仕組みになっている．この調整の役目を果たすのは，タイマー第二次割込み処理ルーチンである．

経過時間を計算するために，2種類の**疑似時計**（psudo clock）が使用されている．6時間疑似時計と24時間疑似時計とである．時間計時機構に6時間に相当する値がセットされるつど，6時間疑似時計にも同じ値をセットする．ただし，計時機構の値はどんどん減じられてゆくが，疑似時計の値は変らない．したがって，6時間以内の経過時間を算出するのは，この疑似時計の値から計時機構の値を減算すればよい．

一方，6時間を越える時間間隔を測るために，6時間に相当する値を6時間疑似時計に挿入するつど同じ値を24時間疑似時計に加算する（ただし，最初の6時間間隔だけは除いて）．なお，24時間が経過するつど，24時間疑似時計はゼロに戻される．ともあれ，TIME ルーチンは6時間疑似時計と24時間疑似時

計の値の合計から計時機構の値を引くことによって，経過時間を算出しているのである．

経過時間が求まると，オペレータがセットした開始時刻にそれを加算することによって，現時点の時刻が求まる．

ところで，計時機構が用いている値は，13マイクロ秒に等しい時間単位である．一方，疑似時計では，26.04秒に等しい時間単位を使用している．そこで，計時機構の値は，26.04秒の単位に変換してから計算に用いられている．TIMEルーチンでは，TIME マクロ命令で DEC オペランドが指定されていると，時刻をパック10進数に変換するし，BIN オペランドが指定されていると符号なしの2進数の値に変換する．ところが，TU オペランドの場合だと変換は不要である．

これらの変換を終えたあと，時刻をレジスター0に，日付をレジスタ1に入れて要求ルーチンに戻すのである．

2）システム/370の TIME ルーチン

システム/370の TIME ルーチンでも，日付はシステム/360の場合と同じように CVTDATE フィールドから得る．TIME ルーチンでは，まず日付のチェックを行ない，そのあと時刻の計算を行なう．時刻は TOD クロックを用いてマイクロ秒の単位で計算される．

TIME ルーチンに制御権が渡るつど，TOD クロックの値を使用して日付の検査が行なわれる．その手順を示したのが，**図8-106** である．

タイマー監視ルーチンの中にある MNIGHT フィールドには，その日の真夜中（つまり，午前0時）に TOD クロックが示すと想定される値が入れられている．TIME ルーチンは，この MNIGHT の値から TOD のその時点の値を引く．その結果が正であると，その日の真夜中にまだなっていないことを意味し，日付は正しいとみなす．逆に，結果が0ないしは負だと，その日の真夜中を過ぎてしまったことになる．そこで，CVT 内の CVTDATE フィールドを1日だけ増やす．また，MNIGHT フィールドの値を24時間分増やしたあとで，同期標識をオンにセットする．日付の差異は2日以上になり得るので，上述の

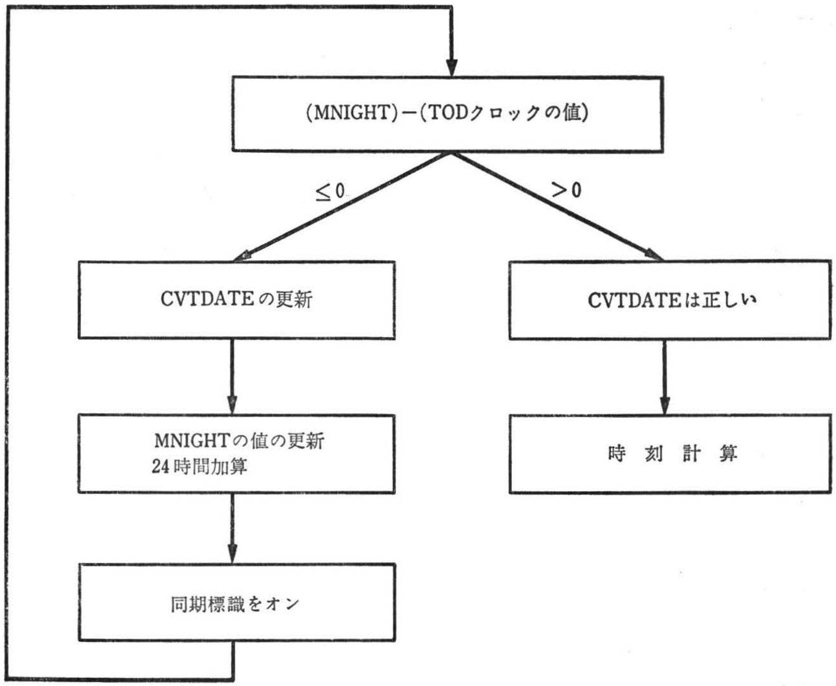


図 8-106 日付けの検査と更新手順

手順は減算の結果が正数になるまでくり返される．

システム/370の TIME ルーチンは，システム/360ルーチンと同様の方法で時刻を計算するのではない．システム/370の TIME ルーチンは，TOD クロックを用いて，下記のアルゴリズムでマイクロ秒の単位で時刻を算出している．

時刻 $= 86.4\times10^9-$(MNIGHT−CLOCK)

86.4×10^9： 1日に相当するマイクロ秒

MNIGHT： その日の真夜中のＴＯＤクロックに相当する値

CLOCK： その時点のＴＯＤクロックの値

(MNIGHT−CLOCK)： 当日の真夜中に至るまでの残りの時間数（マイクロ秒単位）

なお，TIME マクロ命令で MIC オペランドを指定している場合には，ユー

ザーが指定したアドレスの妥当性が検査される．アドレスが妥当でないと，レジスター0はゼロにセットされ，レジスター1に日付が入れられるが，指定された記憶域はそのままの状態にしておいて，レジスター15に戻りコード4をセットして要求ルーチンに制御権を戻す．一方，妥当なアドレスであると，レジスター15には0をセットし，日付はレジスター1に入れ，時刻は要求ルーチンが指定したダブルワードのフィールドに挿入する．なお，この場合もレジスター0はゼロにされる．

TU オペランドが指定されていると，算出された時刻は 26.04166 で除算され，時間単位が調整される．BINないしはDECオペランドと，10000 で割って100 分の1秒の単位に変換される．BIN オペランドの場合には，その結果がそのまま用いられるし，DEC オペランドだと，結果を HHMMSSth 形式に変換してから使用する．

4.　STIMER ルーチンの機能構造

STIMER ルーチンは，STIMER マクロ命令で要求した時間間隔を表示する待行列要素を作りあげ，タイマー待行列の所定位置に挿入する．このルーチンは，必要なら要求された間隔を時間，分，秒の単位からシステムの時間単位に変換する．そのあとで，現存する待行列要素を利用するか，新規に作成するかして，その中に STIMER マクロ命令で指定された情報を入れる．最後に，タイマー・エンキュー・サブルーチンを使用して，タイマー待行列の適当な位置に待行列要素を位置づける．

一方，TTIMER ルーチンは，TTIMER マクロ命令で要請される2種類の機能を遂行する．要求ずみの時間間隔の残り時間を提示する機能と，要求ずみの時間間隔を取消す機能とである．

表 8-28　STIMER マクロ命令の一般形式

〔記 号〕	STIMER	{ REAL〔,タイマー完了エグジット・アドレス〕 TASK〔,タイマー完了エグジット・アドレス〕 WAIT, }	{ ,DINTVL ＝アドレス ,BINTVL ＝アドレス ,TUINTVL ＝アドレス ,TOD＝アドレス }

表 8-29　TTIMER マクロ命令の一般形式

〔記　号〕	TTIMER	〔CANCEL〕

STIMER ルーチンの具体的な機能構造を詳述するに先立って，STIMER マクロ命令と TTIMER マクロ命令の指定内容と機能とを概観しておこう．

前述したように，ある時間間隔をジョブ・ステップ内のタスクに対して設定するためには，STIMER マクロ命令を用いて行ない，間隔内の残り時間をテストしたり，キャンセルしたりするためには TTIMER マクロ命令を用いる．両者のマクロ命令の一般形式を示したのが，**表 8-28** および**表 8-29** である．

STIMER マクロ命令で要求する時間間隔の指定方法には，4 種類ある．つまり，DINTVL，BINTVL，TUINTVL および TOD の 4 通りである．それぞれの指定内容と意味とを**表 8-30** にまとめているので参照して欲しい．

時間間隔内の残り時間を TTIMER マクロ命令でテストすると，残り時間はレジスター 0 に32ビットの符号なしの 2 進数字で戻される．この場合の時間単位は，26.04166マイクロ秒を基準にしている．なお，すでに指定間隔を経過し

表 8-30　STIMER マクロ命令のパラメーター一覧表

パラメータ	指　定　内　容　と　意　味
REAL	時間間隔が連続して引かれてゆくことを指示．
TASK	関連タスクが活動状態にあるときだけ，時間間隔が引かれてゆくことを指示．
WAIT	時間間隔は連続して引かれてゆき，かつその時間が経過してしまうまで関連タスクを待ち状態におくことを指示．
タイマー完了エグジット・ルーチン	指定した時間間隔が経過した直後に制御権を渡すべきルーチンのアドレス．このオペランドが省略されていると，指定時間が経過しても何らの表示も行なわれない．
DINTVL=	時間間隔を収容しているダブルワードの記憶域のアドレスを指定．時間間隔は，HHMMSSth（t は10分の 1 秒，S は百分の 1 秒）の形式でパック10進数表示である．
BINTVL=	時間間隔を収容しているフルワードの記憶域のアドレスを指定．時間間隔は符号なしの32ビットの 2 進数で表示する．時間単位は 0.01秒である．
TUINTVL=	時間間隔を収容しているフルワードの記憶域のアドレスを指定．時間間隔は符号なしの32ビットの 2 進数で表示する．時間単位はタイマーの時間単位，つまり26.04166マイクロ秒を基準にする．
TOD=	時間間隔が満了する時刻を収容しているダブルワード記憶域のアドレス．時刻は，HHMMSSth の形式でパック10進数の表示である．

てしまっていると，レジスター0の内容はゼロにセットされる．

時間間隔を要求するにさいにしては，STIMER マクロ命令で TASK, REAL および WAIT パラメータを使用することによって，時間間隔を減少してゆくさいのやり方を指示することもできる．REAL と WAIT は，関連するタスクが活動状態かそうでないかにかかわりなく連続して引かれてゆくことを意味する．一方，TASK は関連タスクが活動状態のときだけ引かれてゆくことを指示する．また，REAL と TASK では，該当タスクが別の作動可能状態のタスクとの間で御御権の奪い合いが続くが， WAIT を指定すると，時間間隔が経過してしまうまで該当タスクは待ち状態に置かれ，その時間が過ぎたとたんタスクは作動状態に切りかわる．

TASK ないしは REAL を指定した場合には，タイマー完了エグジット・ルーチンのアドレスを指定することができる．所定の時間間隔が完了し，関連タスクが活動状態になったとき最初に制御権が与えられるのが，このルーチンである．このエグジット・ルーチンは要求されたときに，主記憶域に存在している必要があり，レジスターの保存とリストアを行なわねばならない．また，制御権を戻すにあたっては，シジスター14にあるアドレスを使う必要がある．制御権がいったん監視プログラムに戻されてから，主ルーチンのつぎの命令に制御権が渡される仕組みになっているからだ．なお，エグジット・ルーチンのアドレスを指定していない場合には，時間間隔の完了を示すための何らの表示も行なわれない．

プログラムの中のループをテストするさいの時間間隔の使用例を示したのが，**図8-107**である．STIMER マクロ命令で5.12秒の時間間隔をセットし，タスクが活動状態のときだけ時間を減少するようにしている．また， FIXUP と呼ばれるルーチンのアドレスを指示して，時間間隔が完了した時点で制御権が渡るようにしている．ループ自体は BXLE 命令でコントロールしている．

レジスター12の値がレジスター6の値以下である間中ループは続行する．ループが完了すると， TTIMER マクロ命令が実行されることによって，指定された時間間隔の残り時間をキャンセルしてしまう．この場合には，エグジット・ルーチンに制御権は渡らない．しかし，時間間隔が経過してもループがまだ

```
         :
         STIMER   TASK, FIXUP, BINTVL=TIME  時間間隔のセット
LOOP     :
         TM       TIMEXP, X'01'    FIXUP ルーチンに入ったかどうかを
                                   テスト
         BC       1, NG            時間間隔を経過したらループからとび
                                   出す
         BXLE     12, 6, LOOP      処理が未完了だと，ループを繰り返す
         TTIMER   CANCEL           ループが完了すると，残り時間をキャ
                                   ンセル
         :
NG       :
         :
         USING    FIXUP, 15
FIXUP    SAVE     (14, 12)         レジスターの保存
         OI       TIMEXP, X'01'    時間間隔が経過した旨のスイッチをセ
                                   ット
         :
         RETURN   (14, 12)         レジスター内容の復元
         :
TIME     DC       X'00000200'      5.12秒の時間間隔
TIMEXP   DC       X'00'            タイマー・スイッチ
```

図 8-107　インターバル・タイミング（間隔計時）の例

続いている場合だと，FIXUP エグジット・ルーチンに制御権が渡される．エグジット・ルーチンではレジスターの内容を保存し，ループ内でテストされるスイッチをオンにする．FIXUP ルーチンでは，さらにループが成巧裡に終了しなかった旨のメッセージなどを印刷する．そのあとで，レジスターをリストアし，制御権を監視プログラムに戻す．監視プログラムは制御権を主ルーチンに移し，処理を続行する．この時スイッチがテストされると，ブランチが生じループからとび出すのである．

1） タイマー待行列

先述したように，STIMER ルーチンは**タイマー待行列要素**（Timer Queue Element：略して**TQE**）を作り，必要な情報を入れる．必要ならば，指定された時間間隔をシステムの時間単位に変換する．さらに，TQE をタイマー待行列の所定の個所に挿入する．ここでは，システム/360のタイマー待行列とシステム/370のタイマー待行列のそれぞれについて説明してゆこう．

まずは，システム/360のタイマー待行列からみてゆくことにしよう．

タイマー待行列の中のすべての TQE は，各々の時間間隔が満了する時刻の順番に従って配列される．したがって，タイマー割込みが生じると，いつでも最初の TQE が満了した時間間隔を表示していることになる．この TQE が，待行列から取除かれ，どのようなアクションをとるべきかを決定するのに使われる．この間に，つぎの TQE で示されている時間間隔が時間計時機構に入れられ，再び処理手順が開始される．

タイマー待行列上に配列されている TQE で表示されている満了時刻は，6時間サイクルにもとづいている．STIMER ルーチンは，要求された時間間隔を TQE に入れ，タイマー第二次割込み処理ルーチンの中のタイマー・エンキュー・ルーチンを用いて時間間隔を満了時刻に変換して，タイマー待行列にその TQE を入れる．タイマー・エンキュー・ルーチンでは，6時間疑似時計の値から時間計時機構の値を引き，それに要求された時間間隔を加える方法で満了時刻を算出する．つまり，満了時刻に変換する公式はつぎのようになる：

満了時刻＝(6時間疑似時計－時間計時機構の現在値)＋要求時間間隔

このようにして算出された満了時刻が，TQE に入れられるのである．さらにタイマー待行列中の最初にある満了時刻が時間計時機構にセットされる．

たとえば，時間間隔を5時間と要求した場合で，現在の計時機構の値がSET指令から3時間経過していた場合は，

満了時刻＝6－3＋5＝8

となり，タイマー待行列の中の TQE の様子は，**図8-108** のようになる．

一方，システム/370のタイマー待行列もシステム/360の待行列と同じである．ただし，指定された要求時間間隔が1時間を越え，しかも REAL ないしは WAIT タイプの要求を処理するさいには，STIMER ルーチンが TOD クロックを用いる点が大きく異なる（1時間以内の時間間隔で REAL または WAIT タイプの要求とか，TASK タイプの要求はシステム/360の場合と同様の処理が行なわれる）．STIMER ルーチンは要求時間間隔をシステムの時間単位から

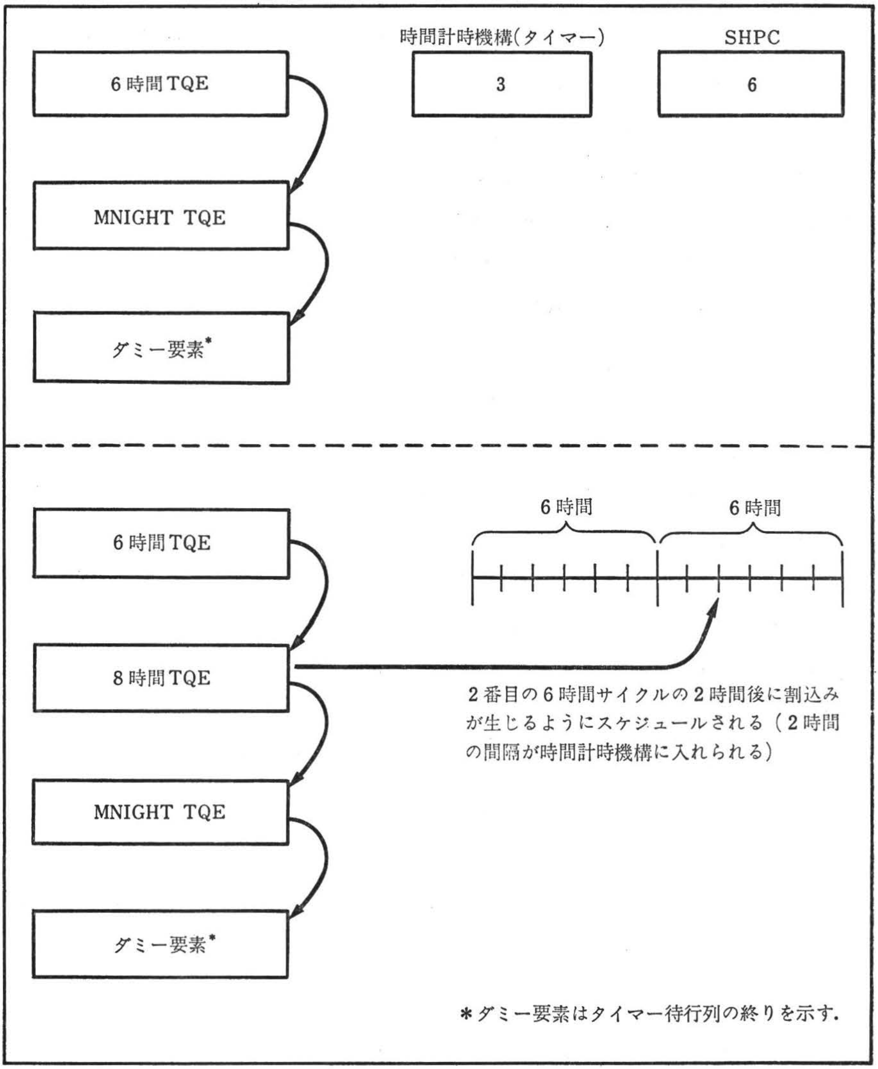


図 8-108　タイマー待行列の様子

1.048576 秒単位に変換し，その値を TOD クロックの現在値に加算する（TOD クロックのビット31は1.048576秒ごとに1ずつ増える．マイクロ秒の値は51ビットで示されるようになっている）．この値は，指定された間隔の満了時刻に

TOD クロックがさし示す値に等しい．したがって，この値を TQE に保存し該当 TQE に対する1時間の時間間隔が経過するつど，タイマー第2次割込み処理ルーチンがこの値を用いる．STIMER ルーチンは，1時間の時間間隔を TQE の TQEVAL フィールドに挿入し，同期化標識をセットする．タイマー・エンキュー・ルーチンでは，この TQEVAL フィールドの値を満了時刻に変換して，その TQE をタイマー待行列に入れる．指定された時間間隔は1時間を越えているけれども，1時間経過すると割込みが生じ，さらに処理が必要かどうかタイマー第2次割込み処理ルーチンで試べるような仕組みになっているのである．

2）間隔値の決定方法

STIMER マクロ命令では，時間間隔を4つの方法で指示することができた．10進形式（DINTVL），2進形式（BINTVL），時間単位（TUINTVL）および満了時刻（TOD）の4種類である．10進ないしは2進形式が指定されると，STIMER ルーチンは，タイマー・エンキュー・ルーチンを用いる前にその値をタイマーの時間単位に変換する．タイマーの時間単位で指定されている場合だと，何らの変換もいらない．

STIMER マクロ命令で満了時刻（TOD）が指定されていると，STIMER ルーチンでは指定された満了時刻から現在の時刻を引くことによって，時刻を時間間隔に変換する．そのあとで，時間間隔をタイマーの時間単位に変換して，タイマー・エンキュー・サブルーチンに引渡す．現在の時刻の算出にあたっては，つぎの公式を用いている：

現在の時刻＝LTPC＋T4PC＋(SHPC－時間計時機構の現在値)

LTPC：オペレータがＳＥＴ指令で与えた開始時刻

T4PC：24時間疑似時計の値

SHPC：6時間疑似時計の値

システム/370においては，STIMER マクロ命令のオペランドで満了時刻（TOD）が指定されていると，STIMER ルーチンは TOD クロックを用いて時間

間隔を決定しているので上述のやり方とは異なる．STIMER ルーチンは，まず満了時刻をその日の開始時刻（0000時）と指定された満了時刻の間の間隔を表わす時間単位の値に変換する．STIMER マクロ命令で TASK ないしは REAL パラメータが指定されており，かつエグジット・ルーチンの指示がないと，TQEがタイマー待行列から除去され，それ以上の処理はなされない。TASK パラメータと一緒にエグジット・ルーチンが指定されていると，タイマー第2次処理ルーチンがその TQE を，IQE を収容した IRB に変更し，ステージ2のエグジット・エフェクターに制御権を渡す．パラメータが REAL で，かつ STIMER マクロ命令を出したのが**開始プログラム**（initiator）でない場合も同様のことがなされる．

開始プログラムが STIMER マクロ命令を出した場合だと，それは30分間待ち制限が経過したことを示している．この場合には，ABTERM ルーチンにブランチして処理プログラムを異常終止させてしまう．

STIMER マクロ命令で WAIT パラメータが与えられると，タイマー第二次割込み処理ルーチンが制御権を FOST ルーチンに渡し，該当 ECB（TQE に収容されている）にポストするよう指示する．このようにして，時間間隔の満了を知らせる．

上記の処理のいずれかを完了したあとで，タイマー待行列の最上位にあるTQE の満了時刻の値を6時間疑似時計に入れる．さらに，満了時刻の値から6時間疑似時計の最新の値を引いた結果を時間計時機構に入れる．そのあとで，この結果から現在の時刻を減算する．現在の時刻は，つぎのアルゴリズムで算出される：

現在の時刻＝(82397－(MNIGHT－CLOCK))×40625

MNIGHT：　その日の真夜中に TOD クロックが示すと想定される値

CLOCK：　TOD クロックの現在値

指定された満了時刻がすでに経過していると（時間間隔がマイナスだと），STIMER ルーチンはその時間間隔に1日分の時間数を加算することによって満了時刻が翌日の指定時刻に生ずるように調整する．

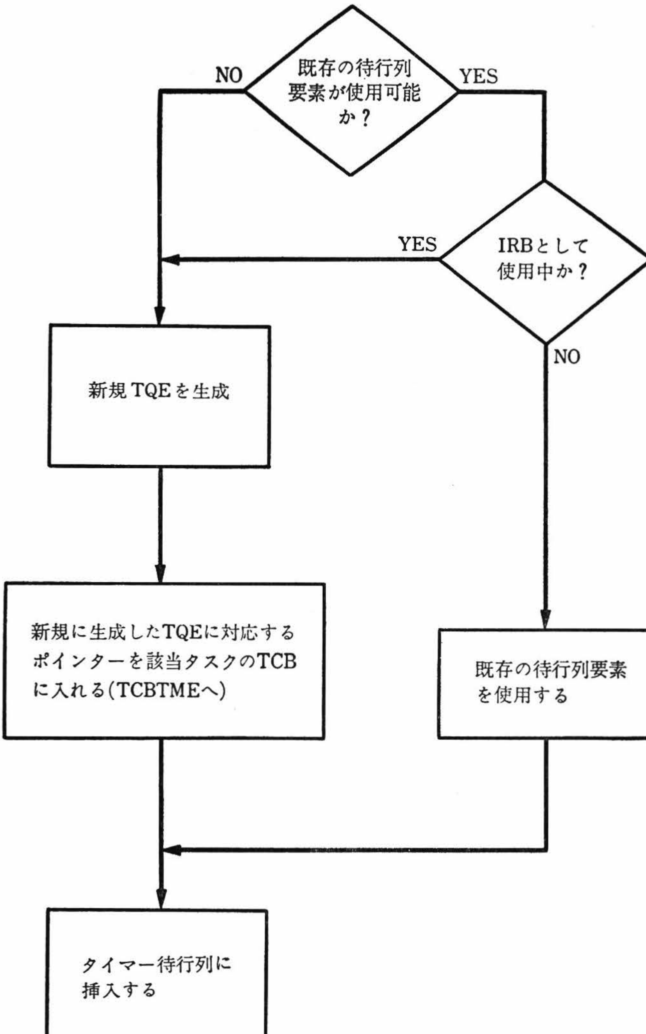


図 8-109　TQE の生成手順

3）TQE の生成手順と TQE の形式

すでに何回も指摘してきたように，STIMER ルーチンは，STIMER マクロ命令で指定された情報をもとに TQE を作りあげる．TQE を STIMER ルーチンが作りあげ手順は図 8-109 に示している．

この図において，既存の待行列要素が使用可能な状態とは，該当タスクに対してすでに STIMER マクロ命令が出されており，かつつぎの3つの条件の時である：

① 該当 TQE の中で指示されている時間間隔が満了している時

② 該当 TQE がまだタイマー待行列の中に存在する時

0	TQEFLGS (インディケータ)	1	TQETCB (TCBのアドレス)
4	ゼ　ロ	5	TQEFLNK (次の待行列要素のアドレス)
8	ゼ　ロ	9	TQEBLNK (先行待行列要素のアドレス)
12	TQEVAL (満了の時刻または残り時間)		
16	TQELHPSW (TQEがIRBとしてサービスを行なっているときに使用されている現PSWの最初1ワード)		
20	TQESAV (TQEが"TASK"から"REAL"に変換するとき，TQESAVを保存する)		
24	ゼ　ロ	25	TQESADDR (処理プログラム用保存域のアドレス)
28	ゼ　ロ	29	TQEEXIT (タイマー用非同期的エグジット・ルーチンのアドレス)
32	TQEWORK		
	SYNC		TQEGRS (TQEがIRBとしてのサービスを行なっているときに使用されている汎用レジスターの保存域)
	TQEECB (TQEがIRBとしてのサービスを行なっているとき，IQE用に使用する)		
96	TQEIQE (STIMERマクロ命令でWAITパラメータを与えたときにECB用に使用する)		

図 8-110　タイマー待行列要素 (Timer Queue Element) の形式と内容

③　タイマー割込みが起こり，TQE が IRB に変えられている時

なお，TQE の形式と内容は，**図8-110**に示した通りである．

5.　タイマー割込みの処理

時間計時機構（interval timer）に入れられている時間間隔が経過すると，**外部割込み**（external interruption）が起り，その結果自動的に**外部第一次割込み処理ルーチン**（external First-Level Interruption Handler；略して**外部 FLIH**）に制御権が渡される．

外部 FLIH は，割込みを起したプログラムに関する情報（主としてレジスターの内容）を保存し，時間計時機構による割込みか別のタイプの外部割込みかを識別し，時間計時機構による割込みだと制御権を**タイマー第二次割込み処理ルーチン**（timer Second Level Interruption Handler；略して**タイマー SLIH**）に移す機能を果たす．

タイマー SLIH は，プログラマーが STIMER マクロ命令で要求した機能を遂行したうえで，時間計時機構に別の時間間隔をセットし，外部 FLIH を経由してディスパッチャーに制御権を渡す．

以上の割込み手順の概略を示したのが，**図8-111**である．

1）　システム/360の場合に遂行する機能

タイマー割込みが生じると，タイマー待行列の最上位の TQE が満了時間間隔を表示している．したがって，タイマー SLIH はその TQE のアドレスを得て，タイマー待行列から該当 TQE を取り除く．そのあとで，どのようなアクションをとるべきかを決定するために，該当 TQE の中の最初のバイト（TQEFLGS）のビット6と7を調べる．その結果，どのようなアクションをとるかを集約したのが**表8-31**である．

タイマー完了エグジット・ルーチン，つまり非同期的エグジット・ルーチンが STIMER マクロ命令で指定されている場合には，つぎのようなアクションがとられることになる

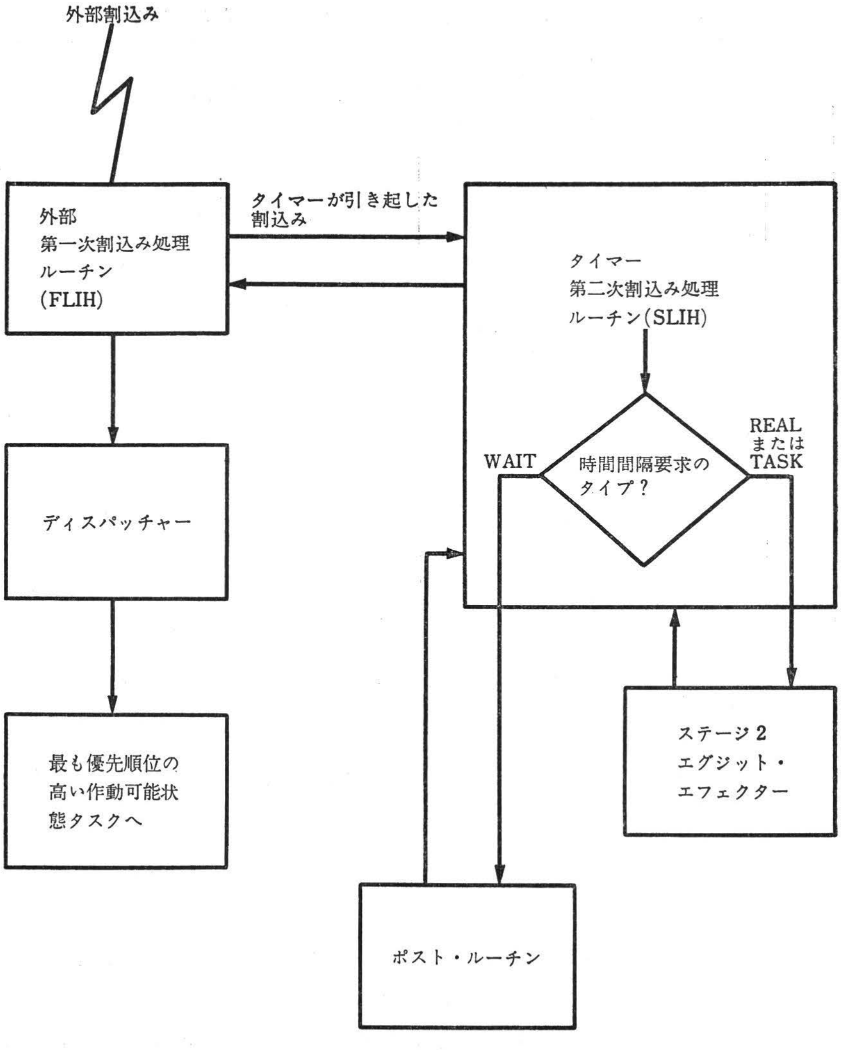


図 8-111　タイマー割込み処理

① **REAL タイプ**： 開始プログラムが STIMER マクロ命令を出していた場合には，30分間の待ち時間が満了したことを意味し，処理プログラムは ABTERM ルーチンによって異常終止させられる．そのあとで，タイプを TASK にかえ，タイマー待行列から該当 TQE を取り除く．一方，開始

表 8-31　TQE のビット6と7の内容とタイマー SLIH がとるアクション

TQE FLGS のビット6と7	表示内容	経過時間の意味	タイマー SLIH がとるアクション
00	STIMER マクロ命令の中で TASK パラメータが使われている.	該当タスクを遂行するのに要した時間	ビット5をチェックする（ビット5が1だと非同期的エグジッドルーチンが指定されていることを意味する）ビット5が1の場合には，ステージ2エグジット・エフェクターに制御権を渡す.
01	STIMER マクロ命令の中で WAIT パラメータが使われている.	時間間隔が計時機構に入れられた時から測定した合計経過時間	POST マクロ命令を実行する (STIMER マクロ命令を出したタスクの実行を再開するためには POST ルーチンが実行される必要がある).
10	満了した時間間隔は，6時間間隔である.	時間間隔が計時機構に入れられた時から測定した合計経過時間	ビット5をチェックする（この場合ビット5が1だと，24時間の間隔が過ぎたことを意味する)．ビット5が1の場合には，日付けを1増やす.
11	STIMER マクロ命令で REAL パラメータが使われている	時間間隔が計時機構に入れられた時から測定した合計経過時間	ビット5をチェックする（ビット5が1だと非同期的エグジット・ルーチンが指定されていることを意味する)．ビット5が1の場合には，ステージ2エグジット・フェクターに制御権を渡す*.

* STIMER マクロ命令を開始プログラムが出した場合だと，TQE を TASK タイプに変換し，制御権を ABTERM ルーチンに渡す.

プログラムが STIMER を出していない場合だと，TQE を IRB に変え，ステージ2エグジット・エフェクターに制御権を渡す.

② **TASK** タイプ：TQE を IRB に変え，ステージ2エグジット・エフェクターに制御権を渡す.

なお，WAIT タイプの場合には，POST ルーチンに制御権を移し，TQE の中の ECB (TQEECB フィールド）をポストする.

以上の処理を終えたあと，新しい時間間隔を6時間疑似時計に入れ，この時間間隔と前の6時間疑似時計の値との差を時間計時機構にセットする.

2）システム/370の場合に遂行する機能

タイマー SLIH は，タイマー待行列の最上位にある TQE を取り除いたうえで，非同期化標識をチェックし，該当 TQE が24時間 TQE かそれともタイ

マー SLIH で特別な処理を必要とするタイプの要求であるかを判定する．もともとの要求時間間隔が1時間を越えるものだと，この標識は STIMER ルーチンがセットしたものである．したがって，この場合にはタイマー SLIH がこの標識をリセットしてしまうことがある．

ともあれ，標識チェックの結果，特別な処理を必要としないことがわかると，システム/360の場合と同じ処理を続行する．非同期化標識がセットされていると，タイマー SLIH は TQEWORK フィールドの中の値から TOD クロックの現在値を引く．その結果がゼロないしは負の場合には，時間間隔が満了したことを意味するのでシステム/360の場合と同じ処理を続ける．結果が正で，しかも1時間未満だと，非同期化標識をリセットし，残り時間間隔を1.048576秒単位からタイマーの時間単位に変換して，該当 TQE の TQEVAL フィールドに入れる．さらに，タイマー・エンキュー・ルーチンを使ってタイマーの時間単位を満了時刻に変換し，タイマー待行列上の TQE に入れる．

計算結果が正で，かつ一時間を越える場合には，TQE の TQEVAL フィールドに再び1時間をセットし，タイマー・エンキュー・ルーチンを使って，タイマーの時間単位を満了時刻に変換し，タイマー待行列上の TQE に入れる方法をとっている．

3） 6時間 TQE と MNIGHT TQE の取扱い

6時間 TQE ないしは MNIGHT TQE で表示されている時間間隔が経過すると，該当 TQE をタイマー待行列に戻す必要がある．6時間 TQE を戻すにあたっては，タイマー SLIH はまずタイマー待行列の中のすべての TQE の満了時刻から6時間を引き，すでに6時間が経過したことを反映させる．また，24時間疑似時計の値が18時間以内だと，それに6時間を加算する．24時間疑似時計の値が18時間を示している場合には0にリセットしてしまう．そのあとでエンキュー・サブルーチンを用いて6時間 TQE を再びタイマー待行列に入れる．

一方 MNIGHT TQE で示される時間間隔が経過した場合には，タイマー SLIH は，MNIGHT TQE をタイマー待行列に戻す前に CVT の中の日付けを

更新する.

6. TTIMER ルーチンの機能構造

TTIMER マクロ命令が出されると, TTIMER ルーチンが機能する. TTIMER ルーチンは, 指定された時間間隔の残り時間の決定と残り時間を取り消す機能を果たす. それぞれの機能について, 以下で説明しよう.

1） 残り時間の決定の仕方

TTIMER ルーチンが残り時間を決定するにあたっては, まず該当する時間間隔を表示している TQE をつきとめる必要がある. そのために, TTIMER マクロ命令を出したときに実行中であったタスクの TCB から該当 TQE のアドレスを得る. TQE が存在しないとか, TQE で表示している時間間隔がすでに過ぎてしまっているときには, TTIMER ルーチンは汎用レジスター0にゼロを入れて, その旨を知らせる. 満了時刻になっていないと, つぎの公式を使って残り時間を決定する：

残り時間 ＝ 満了時刻ー（6時間疑似時計の値ー時間計時機構の現在値）

TTIMER ルーチンの実行中に時間間隔が完了してしまった場合には, 上記の計算結果はマイナスの値になってしまう. この場合には, マイナスの値をそのまま汎用レジスター0に戻す. 残り時間が正の値だと, タイマーの時間単位のまま汎用レジスター0に入れる.

以上がシステム/360 の場合の残り時間の決定手順である. 一方, システム/370 の場合には, TTIMER マクロ命令で非同期化標識がオンの状態の TQE を指定したとき, つぎの手順をふむ点が異なる.

まず, 残り時間の決定にさいしては, TQE の TQEWORK フィールドにある値から TOD の現在値を引く. その結果が, 0ないしは負だと指定時間間隔が過ぎてしまっていることを意味するので, TTIMER ルーチンはレジスター0にゼロを入れる. 算出結果が正だと, タイマーの時間単位に変換してから, レジスター0に入れて, 要求ルーチンに戻す.

2） 残り時間のキャンセルの仕方

TTIMER マクロ命令で CANCEL オプションを使用していると，TTIMER ルーチンは，**タイマー・デキュー・サブルーチン**(timer dequeue subroutine)を用いて，タイマー待行列から対応する TQE を除去する．さらに，TCB の中の TQE ポインターも取り除く．その結果，該当タスクはもはや TQE をもたないことになる．

八．その他のタスク管理機能に関する要約

これまでタスク管理に関する五大機能とそれぞれの面倒をみる監視プログラムの機能構造について詳述してきた．割込み処理機能，タスクの監視機能，主記憶装置の監視機能，内容監視の機能およびタイマー監視の機能がタスク管理の五大機能であり，それぞれの機能を果たす監視プログラム部分が存在する．それぞれの監視プログラム部分は全く独立して機能するわけではない．すでにみてきたように相互に有機的に連係をはかりながら，各々の機能が遂行されてゆく．

これらの機能は，タスク管理の別の機能の援助もうける．エグジット機能とかタスクの終了処理機能，コミュニケーション・タスクの機能などの援助である．コミュニケーション・タスクの機能に関しては，「第７章　ジョブ管理の諸機能とその構造」の「指令処理の内部構造」の個所でふれているので，ここでは言及しない．ここでは，エグジットの手順とタスクの終了処理に関して，それぞれの機能をまとめることにする．両者の機能に関しては，随所で必要のつど断片的に説明してきた．したがって，本節ではそれらを体系的に要約し，タスク管理機能のまとめとしたい．

1. エグジットの処理手順

プログラムでの処理が終了すると，通常 RETURN マクロ命令を出す．RETURN マクロ命令は最終的には「BR 14」命令を実行する．タイプⅠ SVC ルーチンの場合を除いて，「BR 14」命令でブランチした先では「SVC 3」命

令が実行される．SVC 3ルーチンでは，タイプⅠ SVC ルーチンでプログラムがエグジットしたさいの処理の面倒をみる．この SVC 3ルーチンの処理が終るとディスパッチャに制御権を移す．ディスパッチャは，つぎに CPU を使用するタスクを決定し，そのタスクが CPU を使用するための前準備を行なう．そこで，ここではプログラムからのエグジット（つまり，RETURN）とディスパッチャの機能について説明する．

1） タイプⅠ SVC ルーチンからのエグジット

タイプⅠSVC ルーチンが処理を完了して RETURN マクロ命令を出すと，ブランチ命令によってタイプⅠエグジット・ルーチンに行く．タイプⅠエグジット・ルーチンでは，**タスクの切換え**(task switch)が必要かどうか調べる(WAIT, POST などのタイプⅠ SVC ルーチンが，タスク切換えの必要性を認めるとあらかじめ IEATCBP の新 TCB ポインターの値を変えている)．タイプⅠエグジット・ルーチンは，IEATCBP の新・旧 TCB のポインターを比較検査し，両者が異なるときは，タイプⅠ SVC ルーチンがタスクの切換えを要求したことを知る．

タスク切換えの必要があると，タイプⅠエグジット・ルーチンは SVC 用旧 PSW をタイプⅠ SVCルーチンを要求したプログラムの RB に入れる．同様にタイプⅠ SVC ルーチンを要求したプログラムのレジスターを SVC 割込み処理ルーチンのレジスター保存域から TCB のレジスター保存域に移す．そのあとで，ディスパッチャーに制御権を渡す．

逆にタスク切換えを必要としない場合には，SVC 割込み処理ルーチンのレジスター保存域からレジスターをリストアして，タイプⅠ SVC ルーチンの要求を出したプログラムに制御権を戻す（SVC 用旧 PSW を LPSW 命令でロードすることによって）．

2） SPIE エグジット・ルーチンからのエグジット

SPIE エグジット・ルーチンで RETURN マクロ命令を出すと SVC 3 命令が実行される．SPIE エグジット・ルーチンが RETURN マクロ命令を出した場

合に実行される SVC 3 命令は，プログラム割込み FLIH への入口点の直前におかれている．これは，通常の RETURN マクロ命令によって実行される SVC 3 命令が CVT にあるのとは異なる．

SVC 3 ルーチンもタイプⅠ SVC ルーチンではあるが，SVC 用旧 PSW とプログラム割込み用新 PSW の **“次の命令アドレス** (next instruction address)” を比較し，両者が等しいと SPIE エグジット・ルーチンからのリターンであることを知る．SPIE エグジットからのリターンである場合，SVC 3 ルーチンはつぎの処理を行なう：

① PIE の中のフラグ・ビットをリセットする： このビットは，PIE が使用中であること，つまり該当 PIE を使って SPIE エグジット・ルーチンが実行中であることを示している．プログラム割込み処理ルーチンは SPIE エグジット・ルーチンがプログラム割込みを起こしたものとみなして，そのタスクを異常終止させる．いずれにしろ，SPIE エグジット・ルーチンの実行が終了したのであるから，このビットは0に戻しておく必要がある．

② RB の旧 PSW を組み立てる： PSW の左半分は SVC 用旧 PSW から，右半分は PIE の旧 PSW から入手して，RB の旧 PSW を組み立てる．したがって，PIE の旧 PSW の“次の命令アドレス”フィールドの内容をあらかじめ変えておくことによって，SPIE エグジット・ルーチンからの戻り点を変更することも可能である．

③ レジスターの内容を TCB の保存域に入れる．

上記①から③を実行してから，ディスパッチャーへブランチする．

3） ユーザー・プログラムからのエグジット

ユーザー・プログラムの中で RETURN マクロ命令を出すと，SVC 3 命令が実行されて，SVC 3 ルーチンに制御権が渡る．SVC 3 ルーチンでは RB のタイプを調べ，それが PRB であるとタスクを次のレベルで再開させるための準備，および使用ずみのプログラムに対する事後処理を行なうため，つぎに示す

処理を行なう：

① SVC FLIH のレジスター保存域に入っている内容を TCB のレジスター保存域に入れる.

② RETURN マクロ命令をだしたプログラムが該当タスクの唯1つの RB のもとで働いていた場合には，**EOT** (end of task) **ルーチン**にブランチしてタスクの終了処理が行なわれる.

③ RETURN マクロ命令をだしたプログラムが CDE を持っているかどうか調べる（プログラムが SYNCH マクロ命令で実行された場合だと CDE を持たない). CDE を持っていると，それが主 CDE であるかまたは別名に対応する副 CDE であるかを調べる．副 CDE の場合は，それの主 CDE をつきとめる．そのあと，主 CDE の中の CDUSE フィールドから1を引く．主 CDE ではモジュールの性格や状態を示しているので，それらをもとにしてつぎの様な処理を続ける：

● CDE の CDATTR フィールドを検査し，RETURN マクロ命令を出したプログラムが逐次再使用可能なモジュールかどうか調べる．逐次再使用可能なモジュールであると，そのプログラムが使用可能になる状態を待っている要求があるかどうかチェックする．逐次再使用可能なプログラムに対する使用要求は，CDE からの RB チェーンを調べればわかる．つまり，CDE からの RB チェーンが2個以上の RB で構成されていると，該当プログラムの使用完了を待っている要求があることを意味している．この場合には，次の要求を満すために再び該当プログラムの実行を開始する必要がある.

図8-112 を参照して欲しい．RETURN した RB (RB1) を RB チェーンからはずし，2番目の RB (RB2) の待ちカウントをゼロにする．また，2番目の RB の中の旧 PSW の “次の命令アドレス” フィールドを内容監視プログラムの特定アドレスにする．そのあと，新たに作動可能状態になったタスク（TCB2）で CPU を使用できるかどうか判定するためにタスク切換えルーチンに行く.

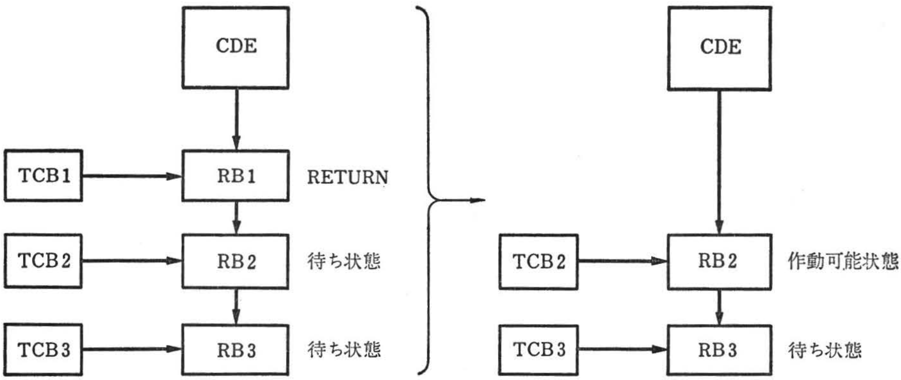


図 8-112　逐次再使用可能なプログラムの要求処理

CDE からチェーンされている RB が1個しかなく，それが RETURN した結果当該プログラムの使用要求が存在しなくなった場合には，CDE に“解放”（release）標識を立てる．“解放”標識が立っている CDE と，それに対応したプログラムとが占める記憶域は，GETMAIN ルーチンで十分なスペースを得ることができないときに利用することができる．

● CDE の CDATTR フィールドを調べた結果，RETURN マクロ命令を出したプログラムが再入可能なモジュールであることが分った場合の処理は該当プログラムが**連係パック域**（link pack area）にあるか，**領域**（region）内にあるかによって異なる．連係パック域にある場合だと何らの処理も必要としない．一方，領域内にある場合には，CDE の CDUSE フィールドの値を調べ，その値がゼロであると“解放”標識を立てる．“解放”標識の意味は逐次再使用可能なプログラムの場合と同じである．

● CDE の CDATTR フィールドを調べた結果，RETURN マクロ命令を出したプログラムの属性が逐次再使用可能でも再入可能でもない場合には，該当プログラム，主 CDE と副 CDE，およびエクステント・リストで占めている記憶域のすべてを解放する．ただし，該当プログラムが LOAD マクロ命令で呼び込まれているときは，スペースの解放は行なわれない．

④ RETURN マクロ命令を出したプログラムに対する RB を TCB の RB チェーンからはずす．また，その RB が TCB の RB チェーン上の唯１つ

の RB である場合は，TCB をディスパッチャーがあつかう TCB 待行列から取り除く．

⑤ 新たに RB チェーンの先頭に位置づけられた RB が待ち状態にあるとき，またはタスクが終了したときは，新 TCB ポインターを0にして，タスクの切換えが必要なことを示す．

⑥ **一時域リフレッシュ・ルーチン**（transient area refresh routine）を経由してディスパッチャーにブランチする．ユーザー・プログラムがRETURNマクロ命令を出した場合だと，一時域リフレッシュ・ルーチンは何もしないで直ちにディスパッチャーに制御権を渡す．

4）ユーザー・エグジット・ルーチンからのエグジット

ユーザー・エグジット・ルーチンで RETURN マクロ命令を出した場合にも，SVC 3命令が実行され，SVC 3ルーチンに制御権が渡る．SVC 3ルーチンでは，IRB の中の RBUSE フィールドから1を引く．その結果，RBUSE フィールドの値がゼロになった場合には，レジスター保存域と IRB が占めているスペースを解放する．そのあとで，一時域リフレッシュ・ルーチンを経由してディスパッチャーに行く．

RBUSE フィールドがゼロでない場合には，直ちに一時域リフレッシュ・ルーチンを経てディスパッチャーに行く．

5）SVC ルーチン（タイプⅡ，Ⅲ，Ⅳ）からのエグジット

タイプⅡ，Ⅲ，ⅣのSVC ルーチンがRETURN マクロ命令を出したときにも，やはり SVC3命令が実行され，SVC3ルーチンが制御権を得る．このあと，SVC3ルーチンはつぎの様な処理を行なう：

① SVRB に保存されているレジスターの内容を TCB のレジスター保存域に移す．また，パラメータ・レジスター（レジスター0，1，15）の内容を TCB のレジスター保存域に入れる．

② 非常駐 SVC ルーチンについては，使用した**一時域ブロック**（Transient

Area Block ; 略して **TAB**) をコントロールするために, つぎの処理をする.

- **一時域制御テーブル** (Transient Area Control Table ; 略して **TACT**) のアドレスと SVRB にある TACT の変位を合計し, 使用した TACT 記入項目のアドレスを得る.
- TACT 記入項目のユーザー待行列を捜し, RETURN した SVRB があれば, その SVRB をチェーンからはずし, 一時域の使用要求を合計したカウンタから1をひく. RETURN した SVRB 以外に, 一時域に入っている SVC ルーチンの使用要求があって, その SVRB がユーザー待行列に入っていると, TACT 記入項目に“使用中”の標識を立てる. 逆に, RETURN した SVRB が, 一時域に入っている SVC ルーチンの唯一の使用要求であったら, TACT 記入項目には“解放”の標識を立てる.

③ SVRB を RB のチェーンからはずし, そのスペースを解放する.

④ 一時域リフレッシュ・ルーチンを経由して, ディスパッチャーへ制御権を渡す.

6) ディスパッチャー

割込みの処理が終了すると, ほとんどの場合, ディスパッチャに制御権が渡る. ディスパッチャーに制御権を渡すルーチンをまとめるとつぎのようになる:

- タイプIエグジット・ルーチン : タイプI SVC ルーチンが実行され, かつタスクの切換えが必要になった場合.
- エグジット・ルーチン : SPIE エグジット・ルーチンが実行されたあと.
- 一時域リフレッシュ・ルーチン : タイプI SVC ルーチン, SPIE エグジット・ルーチンおよび入出力監視プログラム以外のプログラムがエグジットした場合.
- 入出力第一次割込み処理ルーチン.
- 外部第一次割込み処理ルーチン.

- SVC 第二次割込み処理ルーチン： SVC ルーチンを呼び込むため，一時域フェッチ・タスクがスタートした場合．
- 一時域フェッチ・ルーチン： SVC ルーチンの呼び込みが完了したあと．
- ABEND 3 ：ABEND 3 が，終了させるべきタスクを選択したとき．

ディスパッチャーの主要な機能は，いうまでもなく CPU を使用するタスクを選び，そのタスクに CPU を使用させること（つまり，ディスパッチング）である．このほかに，ユーザー・エグジット・ルーチンのスケジュール，タスクおよびジョブ・ステップのタイミング処理を行なう．

以下で，それぞれの機能を要約しておこう．

① ディスパッチング

ディスパッチャーは，IEATCBP から始まる2ワード，すなわち新 TCB ポインターと旧 TCB ポインターの値を比較して，つぎに CPU を使用するタスクをきめる．

この場合，新・旧 TCB のポインターが同じ値を持っているとタスクの切換えは必要ない．新・旧 TCB ポインターが指示している TCB のレジスター保存域からレジスターの内容をリストアーし，TCB がポイントしている RB の旧 PSW を LPSW 命令でロードしてタスクを再開させる．

新・旧 TCB のポインターの値が等しくなく，かつ新 TCB ポインターの値が0でない場合は，目下 CPU を使用しているタスクより高い優先順位をもったタスクが新たに作動可能状態になったことを表わしている．ディスパッチャーは，現在 CPU を使用しているタスクの TCB に浮動小数点レジスターの内容を保存する．汎用レジスターの内容はすでに保存されているので，再び保存する必要はない．この場合，新 TCB ポインターがポイントしている TCB こそが，つぎに CPU を使用するタスクの TCB である．したがって，旧 TCB ポインターの値を新 TCB ポインターの値と等しくする．そのあとで，新 TCB ポインターが指示している TCB の中から汎用レジスターと浮動小数点レジスターの内容とをリストアーする．さらに，そのTCB がポイントしている RB の旧 PSW を LPSW して，該当タスクを実行させる．

一方，新 TCB ポインターの値が0だと，それまで CPU を使用していたタスクが待ち状態に入ったことを意味する．この場合は，TCBTCB フィールドで連鎖されている TCB チェーンを，待ち状態に入ったタスクの TCB を始点に逆探索し，最初に見つかった作動可能状態の TCB をディスパッチする（作動可能状態の TCB とは，その TCB がポイントしている RB の待ちカウントがゼロで，TCB のディスパッチ不能標識がセットされていない TCB のことである）．この様にして見つけた TCB こそは，作動可能なタスクのうちで一番優先順位の高いタスクに対応する TCB である．したがって，この TCB のアドレスを新・旧 TCB ポインターにセットする．そのあと，汎用レジスターと浮動小数点レジスターの内容を，該当 TCB からリストアーし，TCB がポイントする RB の旧 PSW を LPSW 命令でロードして，タスクを実行する．なお，TCB チェーンを逆探索した結果，作動可能なタスクが1つも存在しないことが分ると，疑似 TCB を選び，その TCB が持つ RB から旧 PSW をロードする．これにより，CPU は強制的な待ち状態となる．割込みにより，どれかのタスクが作動可能状態になるとタスク切換えが生じ，作動可能状態になったタスクがスタートする仕組みになっている．

② ユーザー・エグジット・ルーチンのスケジューリング

ディスパッチャーは，ステージ3スィッチがセットされているかどうか調べる．ユーザー・エグジット待行列に記入項目があり，ユーザー・エグジット・ルーチンをスケジュールすることが要求されている場合はステージ3のスィッチがオンになっている．それゆえ，スイッチがオンの場合は，ステージ3エグジット・エフェクターをサブルーチンとして使用し，ユーザー・エグジット・ルーチンの実行をスケジュールする．

③ タスクおよびジョブ・ステップのタイミング処理

ディスパッチャーの処理の結果，タスク切換えが生じたとき，タスクおよびジョブ・ステップのタイミング処理を行なう必要がある．タスクが STIMER マクロ命令を出していた場合，およびジョブ・ステップ・タイミン

グがとられていて開始プログラムが STIMER マクロ命令を出していた場合には，TQE を処理する必要がある．

まずは，タスクのタイミング処理からみてゆこう．

タスク切換えが生じると，それまで CPU を使用していたタスクとこれから CPU を使用するタスクに対してつぎの処理を行なう：

- タスク切換えにより CPU の使用を断念したタスクに対する処理：TASK タイプの STIMER マクロ命令が出されて TQE が作られていた場合は，タイマー第2次割込み処理ルーチンヘブランチする．TQE がタイマー待行列に入っていない場合はディスパッチャーに戻る．TQE が待行列に入っていると，残り時間を計算して TQE へ入れ，待行列からはずす．
- タスク切換えにより CPU の使用権が与えられたタスクに対する処理：TASK タイプの STIMER マクロ命令が出されていた場合は，満了時刻を計算し，その値を TQE に入れたあとで，TQE をタイマー待行列に入れる．TQE を待行列に入れることによって，タイミングがとられるのである．

つぎに，ジョブ・ステップのタイミング処理についてまとめておこう．

ジョブ・ステップのタイミングがとられていて，タスクの切換えがおこるとつぎの処理を行なう：

- CPU の使用を断念したタスクに対する処理：**図 8-113** に示しているような方法で開始プログラム用 TCB をみつける．この TCB が TASK タイプの TQE をもち，かつタイマー待行列に入れられている場合は，待行列から取り除きタイミングをとらないようにする．
- CPU の使用権が与えられたタスクに対する処理：開始プログラム用 TCB が TASK タイプの TQE をポイントしている場合には，タイマー第2次割込み処理ルーチンに行き，ジョブ・ステップのタイミングをとるようにする．

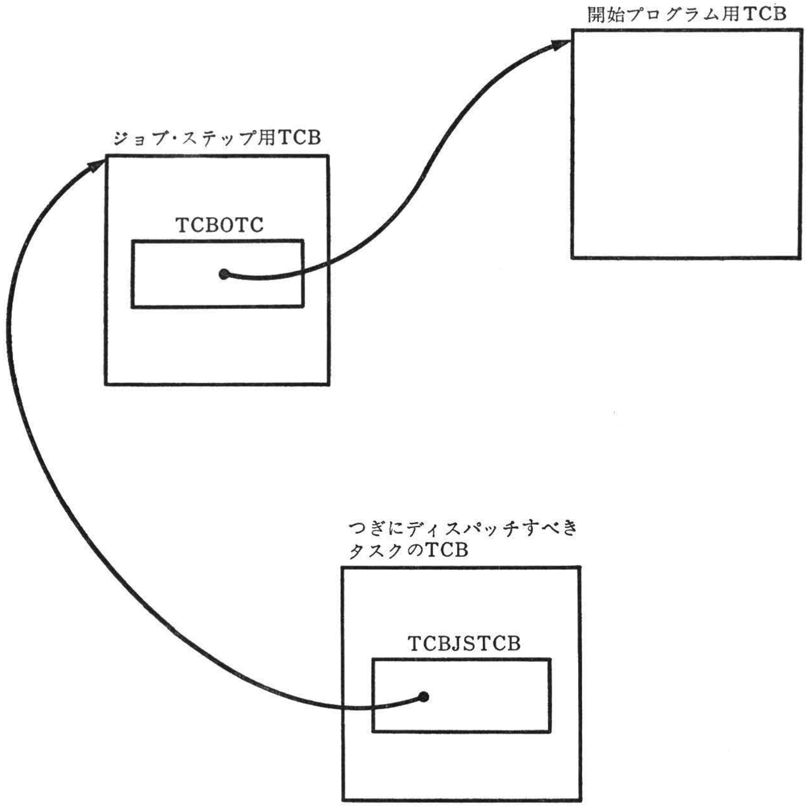


図 8-113　つぎにディスパッチすべきタスクに関連した
開始プログラム用 TCB のつきとめ

2.　タスクの終了処理

タスクが終了すると，そのタスクに割り当てられていたシステム資源を解放する必要がある．解放すべきシステム資源をまとめるとつぎのようになる：

① プログラムおよびプログラムの占めるスペース

② ENQ マクロ命令をかけたシステム資源

③ タイマーからのサービス（サービスが終了していない場合）

④ 応答が完了していない WTOR 命令

⑤ データ・セット
⑥ 主記憶装置のスペース

また，解放される制御ブロックには，つぎのものがある：

① TCB (Task Control Block)
② RB (Request Block)
③ IQE (Interruption Queue Element)
④ QEL (Queue Element)
⑤ QCB (Queue Control Block)
⑥ SPQE (Subpool Queue Element)
⑦ CDE (Contents Directory Entry)
⑧ TQE (Timer Queue Element)
⑨ PIE (Program Interruption Element)

タスクの終了には，**正常な終了**(normal termination) と**異常な終了**(abnormal termination) とがある．正常な終了とは，タスクの最終のプログラムが RETURN した場合であり，異常な終了 (**異常終止**ともいう) とは，修正不能なエラーが起ったためにタスクを終了させる場合である．

ここでは，正常な終了と異常な終了に大別してタスク監視プログラムが行なう機能をまとめることにする．

1) タスクが正常に終了した場合の処理

エグジット・ルーチンが **EOT** (End of Task) の状態を探知すると，制御権を EOT ルーチンに渡す．エグジット・ルーチンが EOT 状態であるとの判定をくだすのは，エグジットしたプログラムに対応した PRB がそのタスクの唯一つの RB である時 (RB の中の RBTCBNXT フィールドにビットがセットされている場合) である．ともあれ，エグジット・ルーチンから EOT ルーチンに制御権が渡されると，EOT ルーチンではつぎに示す処理を遂行する：

① サブタスクの有無の検査

終了したタスクにまだ**消去** (detach) されていないサブタスクがあるかど

うか調べる．消去されていないサブタスクが存在すると，誤りコードをセットしたうえで ABEND マクロ命令を出し，終了したタスクを異常終止させる．一方，すべてのサブタスクが終了し，消去されている場合には，終了したタスクの TCB に**完了コード**（completion code）をセットする．

② PIE，TQE の処理

SPIE マクロ命令が実行されて PIE が作られていると，その PIE が占めている記憶域を解放する．また，STIMER マクロ命令が出され，その結果作成された TQE がまだ残っている場合は，TQE とレジスター保存域のスペースを解放する．

③ まだ DEQ されていないシステム資源に対する処理

TCB の“Enqueue”カウント（TCBQEL フィールド）が 0 でない場合，ENQ がかけられたシステム資源のうち，まだ DEQ マクロ命令が実行されていないものがあることを示している．この場合，EOT ルーチンは誤りコードを準備し，そのタスクに対して ABEND マクロ命令を出す．

④ オペレータ・メッセージおよびオペレータ応答をパージする．

⑤ データ・セットを CLOSE する．

TCB から連鎖されている DEB に対応する DCB を CLOSE する．CLOSE ルーチンの実行中にエラーが生じると，EOT ルーチンは誤りコードを準備して ABEND マクロ命令を出す．

⑥ RETURN したプログラムに対する処理

RETURN したプログラムに対しては，通常のエグジット処理手順と同様の処理が行なわれる．

⑦ LOAD マクロ命令で呼び込まれたモジュールの処理

LOAD マクロ命令でロードされたモジュールで，まだ DELETE マクロ命令が実行されていないモジュールに対する処理を行なう．TCB から連鎖されているロード・リストの“責任”カウント（LLCOUNT フィールド）を，対応する CDE の“使用／責任”カウント（CDUSE フィールド）から引く．その結果，“使用／責任”カウントが 0 でない場合はそれ以上の処理を必要としない．一方，0 だと，次の処理を行なう：

- 再入可能または逐次再使用可能な属性をもったモジュールだと，CDE に解放標識を立てる．
- 再入可能ないしは逐次再使用可能な属性をもたないモジュールだと，CDE，エクステント・リストおよびそのモジュールが占めている記憶域を解放する．

いずれの場合にも，ロード・リストが占めているスペースも解放する．同一ジョブ・ステップ内の別のタスクが該当モジュールを使用している可能性があるので上述のような処理を行なう必要があるのだ．

⑧　タスクが GETMAIN したスペースの解放

タスクが使用している主記憶域は SPQE で表示されており，SPQE は TCB から連鎖されている．これを使用して，GETMAIN したスペースの解放を行なう．スペースが別のタスクとの間で共有されていないと，SPQE とそれがポイントしているスペースを解放する．一方，スペースが別のタスクとの間で共有されていると，SPQE を更新することによってそのスペースの共有が解除されたことを記録するだけで，実際にスペースを解放することはない．

⑨　EOT エグジット・ルーチンのスケジュール

タスクが生成されるとき，ATTACH マクロ命令のオペランドで ETXR が指定されていると，IRB と IQE とが作られ，IQE は TCB から連鎖されている．EOT ルーチンは，ステージ2エグジット・エフェクターを利用して EOT エグジット・ルーチンのスケジューリングを行なう．

⑩　延期されたロールアウト要求があると，それをロールアウト要求待行列から取り除く．

⑪　ディスパッチャーが取り扱う TCB チェーンから，該当 TCB を除去する．

⑫　TCB の“**正常完了** (normal completion)”と“**指名不能完了** (nondispatchable completion)”標識をセットする．

“正常完了”標識は，ジョブ・ステップ TCB に対してだけ意味をもち，

開始プログラムにジョブが正常に終了したことを知らせる．一方，“指名不能完了”標識は，DETACH SVC ルーチンで検査され，この標識がセットされていないタスクを DETACH しようとする場合は，そのタスクを異常終止させる．

⑬ “ATTACH ECB”の POST

タスクが生成されるときに ATTACH マクロ命令で ECB オペランドを指定していた場合には，TCB に入れられている完了コードでその ECB を POST する．

⑭ TCB にに対する処理

タスクが生成されるとき，ECB オペランドも ETXR オペランドも指定していなかった場合は，TCB を親タスクのサブタスク・チェーンからはずし，そのスペースを解放する．

⑮ タスク切換えの必要性を記録

新 TCB ポインターに 0 をセットする．

上記①から⑮の処理を行ったあとで，EOT ルーチンは制御権をエグジット・ルーチンに返す．エグジット・ルーチンは RETURN したプログラムの RB で占めているスペースを解放し，一時域リフレッシュ・ルーチンを経由して，ディスパッチャーに行く．

2） タスクが異常な終了をした場合の処理

タスクが異常な終了をしたときは，つぎの 3 種類のルーチンで処理される：

- ABTERM ルーチン
- ABEND ルーチン
- ABDUMP ルーチン

このうち，ABTERM ルーチンは，ABEND ルーチンの実行をスケジュールする．システム・ルーチンがエラーを発見しても，システムXルーチン自身が ABEND マクロ命令を出すことができない場合がある．このような時，ABTERM ルーチンはそのタスクがつぎに実行する命令が SVC 13 (ABEND SVC)

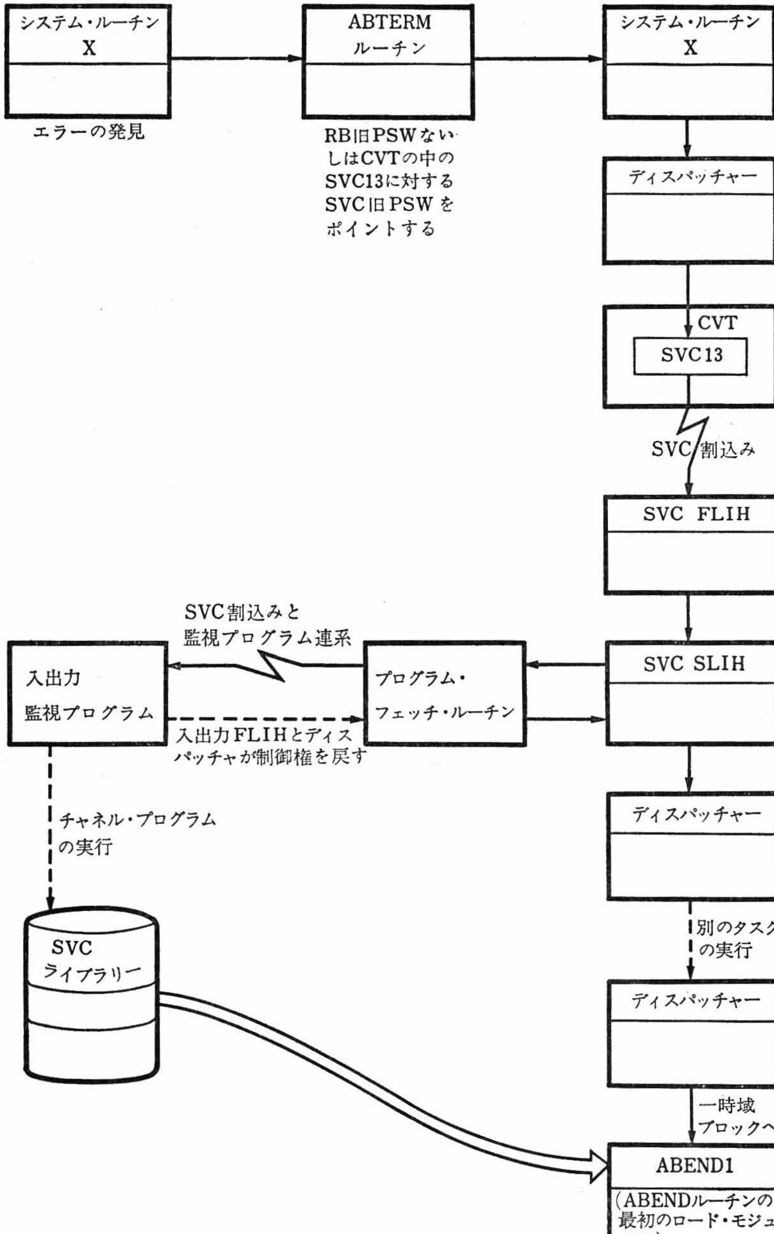


図 8-114　ABTERM ルーチンによる ABEND ルーチンのスケジューリング

であるように前準備を行なう（図 8-114 参照）.

ABEND ルーチンは，終了するタスクとそのサブタスクに割り当てられているシステム資源を解放する.

一方，ABDUMP ルーチンは，ABEND ルーチンまたは SNAP マクロ命令で呼び出され，プログラムと各種制御ブロックをダンプする.

以下で，それぞれのルーチンの働きを要約しておく：

① ABTERM ルーチン

つぎに示すシステム・ルーチンがエラーを発見した場合には，自分自身で ABEND マクロ命令を出すことができないので，ABTERM ルーチンを使ってタスクを異常終止させる：

- 第一次割込み処理ルーチン
- タイプ I SVC ルーチン
- システム・ルーチンが，目下 CPU を使用しているタスク以外のタスクを異常終止させるとき
- SER 1，または MCH
- プログラム・チェック第一次割込み処理ルーチン（この場合だけは，ABTERM ルーチンに直接制御権を渡さないで，ABTERM プロローグ・ルーチンを経て，ABTERM ルーチンへ行く）.

ABTERM ルーチンは，つぎの順序で ABTERM ルーチンをスケジュールする：

i 異常終止させるタスクを決定する.

ii 終了させるタスクの TCB に完了コード，ダンプ・オプションを記録したあと，ABEND ルーチンが使用できる状態にする.

iii TCB でポイントしている RB の中の旧 PSW の“次の命令アドレス”フィールドが，CVT にある SVC 13命令をポイントするようにする.

iv 終了させるタスクのサブタスクを指名不能状態にする.

なお，終了させるべきタスクのタイプによって，ABTERM ルーチンの処理

内容がかなり異なるが，ここでは省略する．

②　ABEND ルーチン

つぎのような回復不能なエラーが発見されると，タスクを異常終止させる：

- 機械チェック・エラー
- 入出力エラー
- プログラム・チェック・エラー
- システム・ルーチン，またはユーザー・ルーチンが異常事態を発見したとき

ABEND ルーチンは，上記のエラーのためタスクが処理を続行することが無意味になったとき，使用しているシステム資源を解放して，別のタスクで利用できるようにする．この ABEND ルーチンは，8つのタイプのタイプⅥ SVC ルーチンから構成され，ABEND マクロ命令を出すことによって実行される．

ABEND ルーチンの処理によって解放されるシステム資源は，EOT ルーチンで解放されるシステム資源と同じである．ただし，そのスケジュールのされ方が異なるだけである．つまり，ABEND ルーチンが SVC 13 命令によってスケジュールされるのに対して，EOT ルーチンはタスクの最後のプログラムが RETURN したときにスケジュールされる．

ABEND ルーチンの実行が終ると，エグジット・ルーチンを経てディスパッチヤーに制御権が渡る．

③　ABDUMP ルーチン

ユーザー・プログラムまたは ABEND ルーチンで SNAP マクロ命令を出すことによって，コア・ダンプをとることができる．SNAP マクロ命令を出すと，SVC 51 命令が実行され，ABDUMP ルーチンに制御権が渡る．なお，ABDUMP ルーチンは9つのタイプⅥ SVC ルーチンと1つの non-SVC ルーチンからなる．non-SVC ルーチンだけは，領域内にロードされ実行される．

第 9 章

仮想記憶システムの概念と構造

一．仮想記憶システム出現の背景

過去10数年の間にコンピュータのハードウェア，ソフトウェアの両分野で驚異的な進歩がみられた．

第１世代のコンピュータのCPUは，ミリセカンドの単位で命令を処理するに過ぎなかったが，現在のコンピュータではマイクロセカンド単位の時代を通り過ぎ，ナノセカンド単位で命令を処理することが普通になった．主記憶装置の容量も飛躍的に増加した．一昔前のコンピュータの主記憶装置の大きさは，せいぜい数千バイトに過ぎなかった．ところが，現在のコンピュータでは，10万バイトを越えるものが普通であり，１千万バイトを越えるシステムも出現している．単に主記憶装置の容量が増大しただけではない．補助記憶装置の容量も目覚しく増大している．磁気ディスク装置が一般化したころの磁気ディスク・パック１個の容量は，百科辞典一冊分の情報量に相当する容量，つまり250万バイト程度であった．ところが，現在はどうであろう．磁気ディスク・パック１個の容量が２億バイトになっており，わずか10年余りで40倍にもなっている．容量が増しただけでなく転送速度も10倍近く速くなった．磁気テープの場合もそうだ．１インチにわずか数百文字といった記憶密度が，今では数千文字になっているし，転送速度も１秒間数万文字であったのが，現在では実に百万文字を越える転送速度のものがある．また，印刷装置，カード読取・穿孔装置においても目覚ましい改良が加えられたし，さまざまな新しいタイプの入出力

装置が開発されてきた．オンライン・システムのように，10数年前にはほとんど存在しなかった適用分野の新しい端末装置については比較のしようもない．

このようにコンピュータを構成する個々の装置の改善だけに目が向けられただけではない．個々の装置の性能向上と併行して，ハードウェア・システムのシステムとしてのバランシングへの挑戦の歴史でもあった．個々の装置の性能向上は，それに見合うだけのシステム全体の性能向上を必ずしも保証しない．つまり，ある特定の装置だけが秀れたものになっても，他のシステム構成要素がその性能を受け入れるだけの作動をしないことには，遊びが生じてしまい，装置個有の性能は見殺しにされかねない．そこで，装置内の速度のアンバランスを克服するための種々のテクニックが採用されてきた．まずは，入出力装置とコンピュータ本体との間での速度のアンバランスが問題とされた．個々の入出力装置の速度の向上を図る一方では，CPU の処理と入出力オペレーションを同時併行的に行なうための試みがなされた．個々の入出力装置に**入出力緩衝機構**（I/O buffer）を設けることによって併行処理を実現したり，サブコンピュータともいうべき入出力チャネルを設けることによって，CPU 処理との完全併行処理と複数個の入出力オペレーションの同時遂行の実現などがその代表例である．さらには，CPU と主記憶装置の間でのデータの転送速度のアンバランスや，命令の解読サイクルと実行サルクルのアンバランスの問題にも目が向けられた．その結果，**高速緩衝記憶機構**（high-speed storage buffer）だとか，命令の同時解析と実行の機構だとかの出現がみられた．さらには，複数個のCPU で1つの主記憶装置を共有することによって，より大きな処理能力と，より高い**使用可能度**（availability）をもつようなシステム，つまり**多重プロセッサー**（multi-processor）が生まれた．

これらハードウェアの性能向上とバランス化と併行して，ソフトウェアの面での改善も行われてきたのは言うまでもない．初期のコンピュータでは，プログラマは機械語でプログラミングすることを余儀無くされた．したがって，コンピュータ内部の仕組みをかなり詳しく知る必要があったし，それにプログラミングがかなり制約された．その後，アセンブリ言語をはじめとした記号言語あるいは，FORTRAN や COBOL で代表される高水準言語，および **IOCS**

(input/output control system）の出現によって，プログラマはコンピュータの機械的な面から解放され，問題解決の仕事により専念できるようになった．しかし，この段階でも，コンピュータはまだ一度に1つのプログラムしか実行できなかったし，各ジョブごとにオペレータが介入し，プログラムやデータのセットを行う必要があった．

ところが，前章までに詳しく説明してきたように，オペレーティング・システムの出現によって，オペレータの介入による遊休時間を最少限にするようなジョブの連続処理が可能となり，オペレーションはより円滑に行なえるようになった．一方，CPU の処理速度の向上と主記憶装置の容量が増加するにしたがって，オペレーティング・システムにも新しい機能が付加されてきた．つまり，CPU，主記憶装置あるいは入出力チャネルなどのシステム資源を，実行中の複数個のジョブに共用させることによって，多重プログラミングが可能になった．さらには，1つのタスクの中で別のサブタスクを発生させることによって多重タスキングを可能とした．また，遠隔地で発生したデータを即座に処理しフィードバックすることのできるようなオンライン・リアルタイム処理を容易にするソフトウェア機能も付加された．そのうえ，多数の端末ユーザにCPU時間を分割して割当て，ユーザーはあたかも独りでコンピュータを占有している気分でコンピュータと会話しながら問題解決にあたることのできるタイムシェアリング・システムのソフトウェアも開発されてきたし，情報資源の1つであるデータの重複を排除し，その効果的利用を図るベータ・ベース・システムの技法も開発されてきた．

このようなコンピュータのハードウェア，ソフトウェアの歴史的発展をふり返ってみると，つぎのような2つの重大な流れがあることに気がつく．その1つは，初期の段階では，個々の装置とか個々のソフトウェアを良くするという比重が高いのに対して，時間が経過するにつれてシステム全体としての性能，つまりトータル・パフォーマンスを重視する傾向が強まってきていることである．このためには，換言するなら個々の装置やフソトウェアの改良を図ることだけでなく，システム資源の遊休状態を極力排除することによって，システム全体としての生産性を高めてゆこうという“がめつさ”の追求である．これは，

前述したようにハードウェアの面からもソフトウェアの面からも改善が志向されてきている．いわば，コンピュータの発展はシステム資源の有効利用の歴史であったともいえるのである．

一方，システム資源の有効利用とからめ合せながらも，別の観点からの追求は，人間にとって使いやすいシステムあるいはより広範囲の分野の問題解決に容易に適用し得る道具としての改良であった．換言するなら，コンピュータが必然的にもっている機械ということからくる冷淡な制約をコンピュータ利用者が厳しく意識することを少なくし，解決すべき問題を主体に思考しうるような環境をハードウェア的にもソフトウェア的にも作り出してゆこうという姿勢である．

これら2つの追求は，まだまだ今後長期にわたって続くであろう．なぜならこれらの問題は完全に飽和点に達するといった性格のものではないからだ．コンピュータという道具がこの世に存在するかぎり問題とされなければならないものといえよう．

ところで，この2つの問題，つまりシステム資源の効率的利用と人的資源にとって使いやすいシステムという問題に，主記憶装置という資源から挑戦したのが**仮想記憶システム**（virtual storage system）といえるのである．

1.　主記憶装置資源の効率的利用の歴史と課題

初期のコンピュータの主記憶装置の容量はせいぜい数千バイトであった．コンピュータで処理する問題が比較的単純であったため，この容量でさほど不便は感じなかった．それに，主記憶装置が高価な代物であっただけに，大きな容量は望めず，記憶容量の制約に合わせてプログラムを分割したり，オーバレイの手法を多用したのも事実だ．ともあれ，初期のコンピュータでは主記憶装置にローディングされるプログラムは，処理プログラムだけであり監視プログラムなんかは全く存在しなかった．処理プログラムとデータとで主記憶装置を占有できたのである．ところで，オーバレイの手法を使わなくて済むようなプログラムの場合には，そのプログラムが実行されている最中に，主記憶装置の何割かは全く無駄になっているのが普通である．たとえば，今8Kバイトの容量

をもったコンピュータを使用していると仮定しよう．作成したプログラムが全て８Ｋバイトの大きさであれば，ローディングされたプログラムで何時でも記憶域全体が占有されていることになるから，まあ無駄はないといえよう．だが全てのプログラムが８Ｋであるなんていうことは現実的ではない．ある種のプログラムは２Ｋそこそこのものもあるし，４Ｋ程度のものもあろう．これらのプログラムが実行されているときは，主記憶域の半分以上が無駄になっているのである．それに，プログラムのある部分は例外処理のためにしか使わないにもかかわらず，他の頻繁に使われている部分と同様に記憶域を占めているのである．記憶装置の効率的利用という観点からすれば，実に勿体ないことである．しかし，圧倒的多くのユーザがこれを当然のこととして是認してきたのである．

オペレーティング・システムが使われる時代になっても，その初期の段階にはこの問題は放置されたままであった．初期のオペレーティング・システムでは，**図 9-1** の左側で示しているように単一区画方式であった．換言すれば多重プログラミングの考えは採用されていなかった．この場合には，主記憶域の有効利用という観点だけからみれば，監視プログラムが一定の記憶域を占めてしまうことさえ無駄なことのように思えた．いずれにしろ，主記憶装置の何割かが未使用の状態におかれる問題も，使用頻度が極めて低いプログラム部分にも

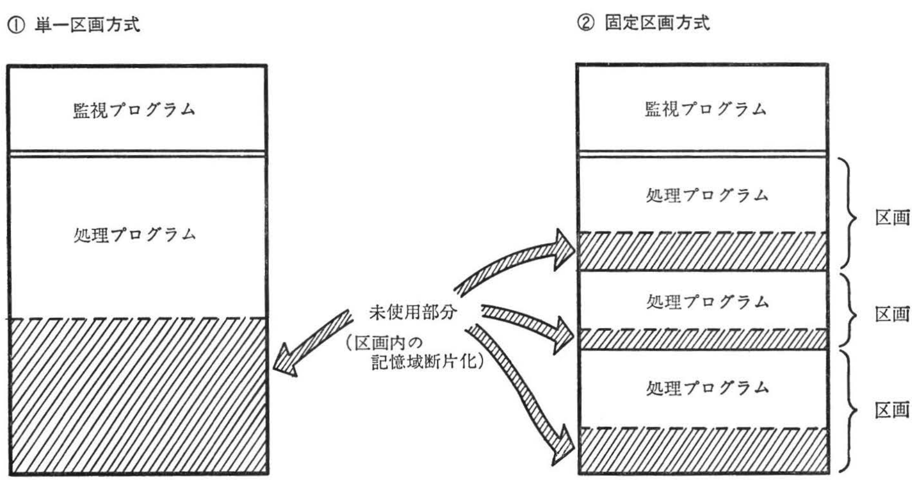


図 9-1　単一区画方式と固定区画方式

公平にスペースを与えてしまうといった問題もそれほど解決はされなかった．

ところが，監視プログラムが主記憶装置のかなりを占めていることに関しては比較的早くから問題視されていた．その機能を制約することなく，記憶域は経済的に使いたいという願いは強かった．その結果，使用頻度の高い監視プログラムとそうでない監視プログラムの主記憶装置の利用の仕方を変えることが追求された．常駐ルーチンと非常駐ルーチンの考え方である．使用頻度の低いルーチンは，そのルーチンを必要としたときだけ記憶域を使わせようとしたのである．この方法が採用されることによって，オペレーティング・システムの機能を大幅に増やしたにもかかわらず，それに比例するだけの大きな記憶域を監視プログラムにさかなくて済んだのである．

処理プログラムの中でオーバレイ手法を使用することは早くから行なわれていた．しかし，この手法ではプログラム作成時にプログラムの実行順序を見通し，それに合わせてオーバレイ構造を計画する必要があった．処理する仕事が複雑になり，条件によってさまざまな例外処理を行なわなければならないような場合だと，なかなかうまく対処しきれない場合もしばしばである．そこで，監視プログラムにおいて，使用頻度の低いルーチンを必要のつど，ダイナミックに呼び込んで処理を行わせるのと同じような考えが，処理プログラムにも採用された．LINK，XCTL，LOAD あるいは ATTACH などのマクロ命令を用いて，必要な処理ルーチンをダイナミックに呼び込む方法である．これによって，使用頻度に応じた主記憶域の利用という問題はある程度解決したのである．

ところで，主記憶装置の容量が大きくなり，CPU の処理速度が速くなる一方，システムの生産性が強く意識されるようになってくると，必然的に多重プログラミングの方向が目指されるようになった．その結果，まず固定区画方式の多重プログラミングの方法が採用された．

固定区画方式とは，システム生成時あるいは IPL 時に処理プログラムを実行する記憶域の大きさを固定してしまうやり方である．場合によっては IPL 後にオペレータが**指令**（command）を使って区画の大きさを変更することは許される．しかし，ジョブないしはジョブ・ステップの実行時点でそれぞれが要求する記憶域の大きさに合わせて動的に区域を変更することはできない．した

がって，**固定区画**と呼ぶのである．この方式だと，確かに多重プログラミングが遂行できることから全体のスループットの向上は約束される．しかし，記憶装置の有効利用という観点からすると，さきの単一区画方式の場合の問題はそのまま残されている．区画が固定されていることから，各区画に未使用の記憶域が散在することになる．このことは，図 9-1 の右側に示した通りである．このように記憶装置の未使用の場所ができることを**記憶域の断片化**（storage fragmentation）という． このように**各区画**（partition）の中に断片化が生じることを，特に**区画内断片化**（intra-partition fragmentation）と呼ぶ．

ともあれ，各区画内の記憶域に無駄が生じるだけに，単一区画方式の場合よりも始末が悪い．それに，ジョブの性格やプログラムの大きさの関係から，ある特定の区画が一時的に遊んでしまう状態が生じることもある．つまり**区画間の断片化**（inter-partition fragmentation）である．また，区画の数を増やすと，それだけ個々の区画は小さくなり，その結果大きなプログラムを実行することができなくなってしまう．

固定区画方式だと，主記憶装置の管理は比較的単純である．それだけ監視プログラムの中の**主記憶装置監視プログラム**（main storage supervisor）の機能も簡単であるが，これまでみてきたような主記憶装置の効果的利用の問題が残されるのである．そこで，第 8 章の「タスク管理の諸機能とその構造」の「主記

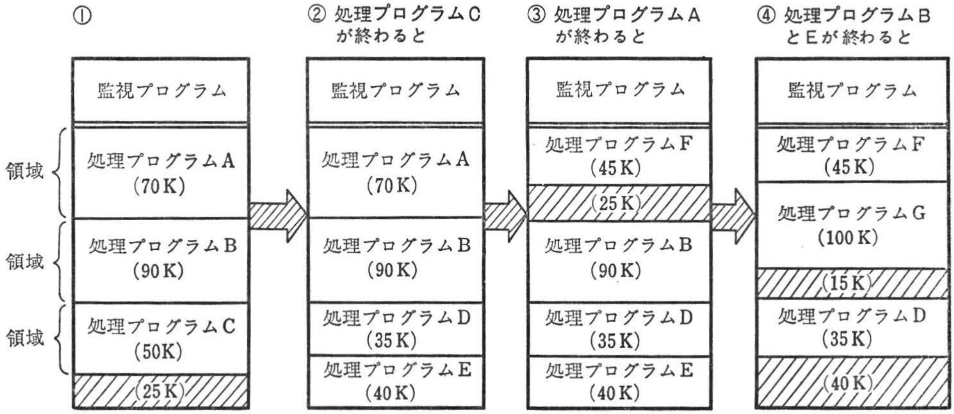


図 9-2 可 変 区 画 方 式

憶装置監視プログラムの機能構造」の個所で詳述したような複雑な管理の仕方が出現するのである．これが，可変区画方式である．つまり，ジョブやジョブ・ステップで要求する記憶域に合わせて，そのつど動的に**記憶領域**（region）を割り振る方法である．

可変区画方式の記憶域の割当ての様子を示したのが**図 9-2** である．この方法だと，その時々に合わせた記憶域を領域として要求するので，領域内での未使用の部分は，固定区画方式に比較して大幅に改善される．それに，大きなプログラムを実行する場合には，それに見合った大きさを要求すればよいので，作成するプログラムの大きさに関する制約は少なくなろう．しかし，図 9-2 で示しているように領域と領域の間にしばしば断片化が生じる．各々のプログラムは必ず連続したスペースを要求するので，要求を満足するだけの空きがある場所から必要な領域を割り振るので，領域と領域の間に無駄が生じるのは避けられないのである．また，**図 9-3** の左側の図で示しているような状況も生じる．断片化の総容量が要求する領域の大きさを越えるものであっても，連続した場所でないことから，そのジョブを開始することができない．そこで，ロールアウト／ロールインの機能が活用される．つまり，該当ジョブの優先順位が実行中のものより高い場合だと，実行中のタスクで占めている領域の内容を一時的

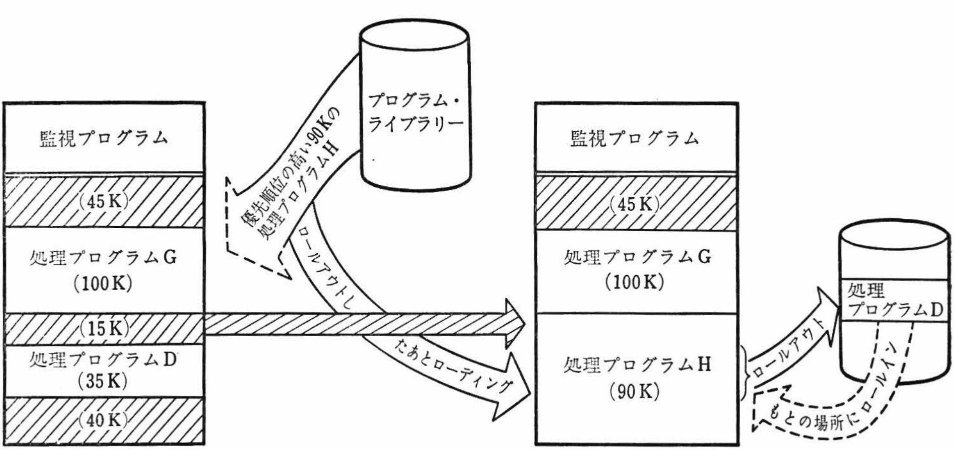


図 9-3　ロールアウト／ロールインによる解決

に補助記憶装置上にはき出し，優先順位の高いジョブに借し与える方法である．

これによって，優先順位の高い仕事はスムーズに処理される．しかし，問題がないわけではない．1たんロールアウトされたタスクは，もと占めていた場所が空かないかぎりロールインできないのである．たとえ，別の記憶域に十分な空きができたとしても，その場所にロールインして実行することは動的再配置に関するハードウェア機構がないことには，かなりむずかしいことである．可変区画方式は，固定区画方式に比較して記憶装置の有効利用とプログラミング上の制約解除という視点からみると，かなりの改善にはなっているが，根本的な解決という面からはほど遠いのである．

そこで，単にソフトウェア面からの解決方法を追求するのではなく，ハードウェア面での機能追加とからみ合せて解決する方法が検討されるようになった．その結果出現したのが，**主記憶域コンパクション**（storage compaction）とプログラムの**スキャター・ローディング**（scatter loading）の2つの方法である．

主記憶域コンパクションの方法を採用した場合には，プログラムを主記憶装置にローディングするさい，特別なローダが主記憶装置の使用可能な記憶域を捜すのであるが，該当するプログラムを収容するに十分な連続した場所がないときは，すでに記憶域にあるプログラムを適宜移動させて空間をつめたのちに，プログラム・ローディングを行なう．これによって，記憶域の断片化の回避を図っているのである．

図9-4をみていただきたい．現在，処理プログラムA，B，C，D，Eが実行されていると仮定しよう．一定時間経過して，プログラムA，C，Eが終了したとする．この時，主記憶装置は50K，100K，50Kと飛び右のような状態で未使用の場所が発生する．ここで，180Kの処理プログラムをローディングする必要が生じたとする．プログラムはすべて連続した場所に入らなければならないので，主記憶域のコンパクションが実行され，200Kの連続したエリアが得られる．このようにして，十分な記憶域が確保されたあと，プログラムKがローディングされる仕組みになっているのである．

必要に応じてすでに実行中のプログラムを再配置するのであるから，記憶されているプログラムの番地は絶対番地であることは許されない．各々の命令が

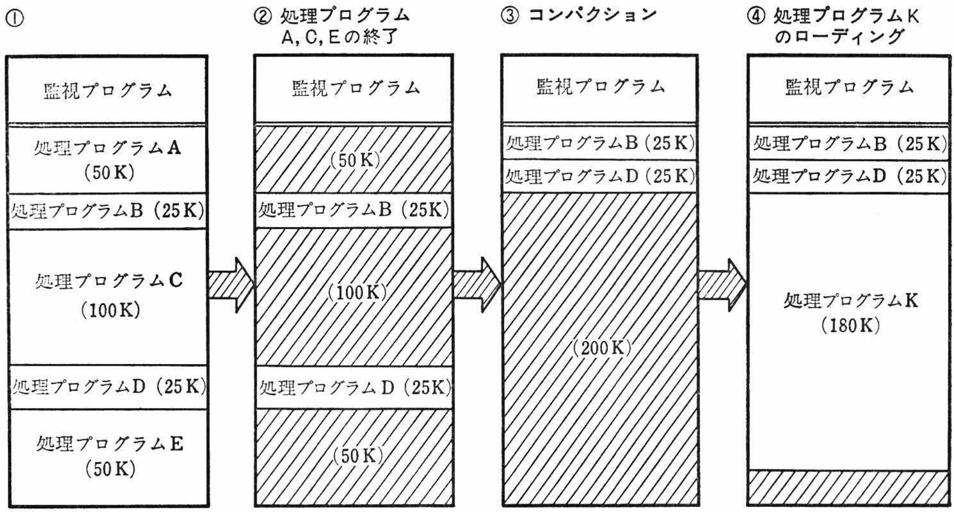


図 9-4　主記憶域コンパクションによる解決

実行されるさいに，相対番地を絶対番地に変換するようなハードウェア的仕掛が必要であろう．このために，再配置用のレジスタ的なものが CPU に組込まれていることが前提となる．

ともあれ，主記憶域のコンパクションの方法だと，記憶域の断片化の問題はかなり解決するし，ロールアウトされたプログラムのロールインは，元の記憶域に制約されることもなくなる．そのプログラムを収容するに十分な記憶域であればどこにロールインしても，支障なく実行できるといえる．しかし，この方法に全く問題がないわけではない．実行すべきプログラムの全体がローディングされるので，使用頻度の大小によって記憶域の占有時間をコントロールするまでには至らないのである．

一方，スキャター・ローディング方式は，ローディングすべきプログラムは連続した記憶域に収容されなければならないといった固定観念を打破するやり方である．**図 9-5** に示しているように，記憶装置内の空きの場所をみつけて，どんどんローディングする方法をとる．したがって，1つのプログラムがあちこちに不連続の状態で分散して記憶される．この方式だと，プログラムがロー

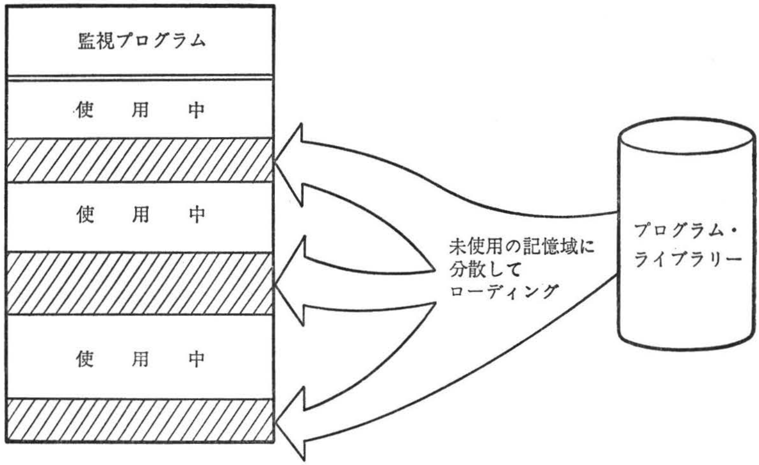


図 9-5　スキャター・ローディングによる解決

ディングされてみて，はじめてその記憶番地が定まるのであるし，物理的な番地は連続しないことになるので，当然のことながら何らかのハードウェアの援助で正確な番地をつかむ必要がでてくる．つまり，プログラムが物理的にどのような位置に分散されていようとも論理的には連続して取扱うことができるようなアドレス変換機構が必要なのである．

ともあれ，スキャター・ローディングにより，主記憶装置はまんべんなく使用でき，効率的な主記憶装置の利用が期待できる．また，必要なプログラム部分を必要のつど空きの場所をみつけだしてローディングするということも実現できよう．ただし，プログラムの分割をどのようにするかの問題とかかわりのあることだが，記憶域の断片化が完全に解消しきれない面が残る．

以上みてきたように，主記憶装置の効率的使用に関しても，さまざまな工夫がなされ，大幅な改善がなされてきたのである．しかし，十分に解決しきれていない点もある．要は，つぎに示す3つの課題が解決されることによって，主記憶装置の本来的な効率的利用が実現されるといえるし，次節以降で考察する仮想記憶装置はまさにこれら課題に挑戦しているのである：

1.　主記憶装置の断片化を極力防止する．この点に関しては，記憶域のコン

パクションやプログラムのスキャター・ローディングの方法でも大幅に改善できることはすでにみた通りである．

2. プログラムは，その実行中の必要時点でのみ主記憶装置を用いることが望ましい．たとえば，プログラム全体をローディングすると10万バイトを必要とするプログラムでも，特定データの処理には2万バイト程度しか使用していないかも知れない．この間に使用されない8万バイトの記憶域は，他の仕事に解放する方がはるかに望ましい．

3. 主記憶域は，活動中のジョブの間で動的に活用されることが望ましい．しかも，必要なときに必要なだけの記憶域が使えるのが望ましい．換言すれば，仕事を遂行するうえで必要な記憶域は待たずに割り振られることが必要だし，出来るだけ多くのジョブが同一記憶域を共有して実行できることが必要である．

2. プログラム設計上の制約解消のための課題

多重プログラミングやタイムシェアリングの出現によって，システム資源を複数のタスクで共同利用するということは，もはや当り前になっている．共同利用される資源の中に主記憶装置が含まれることはもちろんである．しかし，限られた資源である主記憶装置をいくつかのプログラムで共用するということは，逆にそれを共同利用するプログラムにとっては，何らかの制約が課されるということも事実であり，このことはプログラマがプログラムを作成するさいの問題として残される．

ここで，資源の1つとして，主記憶装置の記憶域を例にとりあげて，共同利用する場合にいかなる問題が生じるかを検討してみよう．

たとえば，FORTRAN 言語で300×200の行列を取扱かうプログラムを作成する場合，次のように書けば，もっとも簡単にプログラミングでき，その処理も早くなろう：

```
DIMENSION A (300, 200)
        ⋮
```

しかし，これを処理するコンピュータの主記憶装置が256Kバイトしかないと

したら，この方法は再検討する必要が生じよう．つまり，行列の１つのエントリーに４バイト要するとしたら，上記の行列全体に必要な主記憶域は，300×200×4＝240000 バイトとなる．全体256Kバイトのうち，この行列だけのために240K バイトもとられてしまうと，処理プログラムの部分やオペレーティング・システムの制御プログラムの部分にわずか16Kバイトしか残らない．これでは実用的でない．そこで，プログラマはプログラム構造の設計変更の必要性に迫られる．あるプログラマは，入出力ステートメントを余計につけ加えて，行列用のデータを磁気ディスクから何回かに分割して読み込むかも知れない．しかし，このためには別のプログラムで行列用データをあらかじめ磁気ディスクに貯えておくという余計の仕事をやっておかねばならない．また，別のプログラマはプログラムの構造を変えてオーバレイ構造にし，何んとか限られた記憶域に収容するということをやるかも知れない．この場合にも，プログラムを分割する必要があり，そのためにプログラム論理が多少複雑になるということは避けられない．これらの努力は，問題を解くという目的からすると，無い方がよいという類のものであり，プログラマにとっては余計な仕事である．もし十分な主記憶域があれば，これらの努力は全く不要なことでもる．

そこで，仮想記憶装置という概念上の記憶域を想定し，プログラマが実際の主記憶装置に制約されることなく，この概念上の仮想記憶域に書くようにするということが考えられるようになったのである．もちろん，仮想記憶装置上のスペースの大きさにも制限があるが，実際の主記憶装置のそれよりもはるかに大きいものが想定できるのであり，プログラマにとっての記憶域上の制限を大幅に緩和することが可能である．

ともあれ，問題解決のためのプログラムを設計するにあたって，解決すべき問題だけに努力が傾注でき，また環境の変化によって生じるプログラム変更の負担をできるだけ少なくするといったことに関連して，主記憶装置の管理という観点から提供される課題は，つぎの４点に集約できる．しかも，仮想記憶システムは，これらの課題にこたえるように設計されているのである：

1. ユーザーは，プログラムが実行される主記憶装置の大きさや監視プログ

ラムが占める記憶域の場所と大きさに制約されることがない方がよい．主記憶装置の大きさに気を取られることを最少限にし，解決すべき問題に重点を置いて，システムやプログラムの設計がなされることが望ましい．

2. プログラマがオーバレイ構造や動的処理構造などわずらわしいテクニックを応用しなくても，プログラム・コーディングができるだけの主記憶域が与えられることが望ましい．これまでの傾向だと，たとえ主記憶装置の容量が飛躍的に増大しても，多重プログラミングのためにいくつかの区画や領域に区切られるので，プログラム設計にあたっては以前と同じように区画や領域の大きさにしばれてきた．このために依然として，いや前以上に，プログラムの分割やオーバレイが多用され，複雑なプログラム構造になってきている．その結果，問題解決に余分な手間がかかることになる．さらに見逃すことのできないこととして，完成プログラムの保守が極めて面倒くさくなっている．プログラムは，単純な論理構造で組立てる方が，その作成も容易だし，保守も簡単になる．

3. 主記憶装置のサイズよりプログラムのサイズの方が大きい場合でも，そのプログラムを主記憶装置に合わせる仕事は，プログラマが行うのではなく，オペレーティング・システムが自動的に行うほうが望ましい．

4. オペレーティング・システムが主記憶装置の管理を行い，主記憶装置を拡張したような場合でも自動的にパフォーマンスをあげるようにしてくれることが望ましい．たとえば，主記憶装置の大きさに合わないためやむを得ずオーバレイを用いてプログラムを書いた場合，新たに主記憶装置のサイズを大きくした結果，そのプログラムを設計し直したり，コーディングし直したりする必要性が生じる．このような手直しをしなくても，オペレーティング・システムが自動的にプログラムをオーバレイし，追加された記憶域を効率的に使えるようにしてくれることが望ましいのである．

3.　仮想記憶システムへの必然性

過去のオペレーティング・システムに与えられた主要課題の1つである「主記憶装置の使用効率の向上」という問題と，プログラミング上の大きな制約事

項であった「主記憶装置の大きさからプログラマを解放する」という２つの問題が動機づけとなって，“より優れた記憶装置の管理”への追求がなされてきたわけである．システム資源の中で，特に主記憶装置はその容量に限界があるので，高速で高価な主記憶装置と，相対的には低速で安価な補助記憶装置の量的配分について常に妥協を強いられていただけに，主記憶装置の管理の仕方はオペレーティング・システムを計設するうえでもかなり重要な問題として絶えず配慮が払われてきたのも事実である．

この問題は，単に安く大きな主記憶装置が提供されれば解決するといった単純なものではない．確かにプログラマが利用することのできる記憶域は増えるが，主記憶装置の使用効率の問題は依然として残されるからである．1960年以降10数年にわたる研究と試行の結果，現在では仮想記憶の概念が記憶域管理方法として非常に優れた１つであることが一般的に認められるようになった．この仮想記憶の概念とは，プログラムのために使用できる記憶域(アドレス空間)を，コンピュータに実際に存在している主記憶装置（つまり，実記憶装置）の大きさの制約から独立させようというものである．プログラマには，ハードウェアの構造上許される最大のアドレス空間，つまり概念上の記憶域をあたえてプログラミングさせ，プログラムの実行時点においてある種の**マッピング機構**(mapping device）によって，実記憶装置を割り付けていこうという構想である（**図9-6**参照）．なお，後述するように，**アドレス空間**(address space)から実記憶装置上のスペースへの割付けの単位のことを**ページ**(page)と呼んでいる．

ページは通常２Ｋバイトあるいは４Ｋバイトの固定長である．当然実記憶装置の大きさには制限があるので，一度に処理プログラム全体を収容することはむずかしい．したがって，実記憶装置に収容できないプログラム部分は，仮想記憶装置を反映した補助記憶装置上に記憶されていて，そのプログラム部分が参照された時点でページ単位に実記憶装置に読み込まれる．この場合に，前述のマッピング機構と制御プログラムの働きで実記憶装置上の場所が割り付けられるのである．実記憶装置上にページを収容するためのスペースが空いていないときは，最も使用頻度の低いプログラム部分が補助記憶装置上に書き出され（これをページアウトという），他のページを読み込むために使われるような

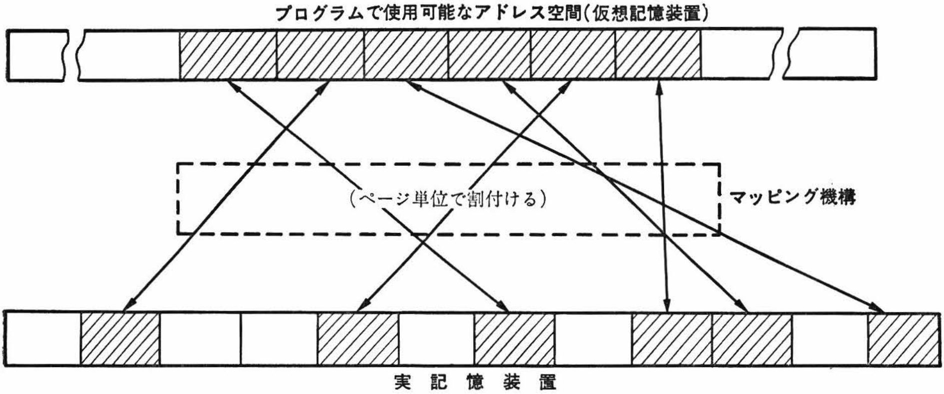


図 9-6　実記憶装置上の場所の割付けの概念図

仕組みになっている．このように，必要とするプログラム部分を要求された時点に実記憶装置に持ち込む一連の操作のことを，一般に**要求時ページング**(demand paging) と呼んでいる．

ともあれ，仮想記憶装置と要求時ページングの手法とを採用することにより，実記憶装置上での断片化がなくなり，使用頻度に応じた記憶域の利用の仕方が実現できるのである．これによって，「プログラマにより大きなアドレス空間を与え」,「実記憶装置の使用効率の向上」という二大課題に対処できる可能性が生れるのである．

さて，実記憶装置の使用効率の向上の問題であるが，これは見方を変えるとプログラムの実行に必要な実記憶装置の割り当てをできるだけ遅らせることによって，その向上を計ることができると言えよう．そこで，この観点から他のアドレス指定機構と比較してみることにする：

①**絶対再配置** (absolute relocation)： プログラムのアセンブル，コンパイルあるいは連係編集時に実記憶装置の場所を決めてしまうやり方である．目的モジュールないしはロード・モジュールの中に含まれている番地が絶対的な番地であり，その番地で示される場所にしかプログラムはローディングできない．したがって，別の区域でそのプログラムを実行するためには，ローディ

ング・ポイントを変更してコンパイルをやり直すか，または連係編集をやり直す必要が生じる．この方式だと，融通性のある多重プログラミングを実現することはむずかしい．

②**静的再配置**（static relocation）：静的再配置は，多重プログラミングでどの区画においてもプログラムを実行できるようにするために採用された手法である．プログラムが連係編集されたあとでも，プログラムの中に含まれる番番は相対番地であり，プログラムがローディングされる時点に実記憶装置の番地を決定する．したがって，プログラムをローディングするさいに空いているところならどこにでもそのプログラムを読み込めるのである．これによって，柔軟性のある多重プログラミングは可能となったが，記憶域の断片化を解消することにはあまり役立たなかった．

③**動的再配置**（dynamic relocation）：プログラムの実行時点で実記憶装置上のアドレスを決めようとするものであり，初期の**再配置レジスタ**（relocation register）方式では，レジスタに収容されているプログラムのロード・ポイントとプログラムの中の命令で指定されている相対番地を加えることによって実際番地を得た．この方式でも，プログラムをローディングする時点で実記憶装置上の連続した記憶域を割り当てる必要があるので，やはり記憶域断片化の問題が残る．ただ，プログラムをいったんローディングして，実行している最中に，再配置レジスタの内容を変えることができるので，プログラムの記憶場所を移動し，その移動した結果を再配置レジスタに反映させることによって，記憶域コンパクションが実現できる．

このほかにも，動的再配置の手法としてさまざまな方法が研究されているが，現在一般化しているのが，通常**動的アドレス変換機構**（dynamic address translation；略して**DAT**）と呼ばれているマッピング機構を利用した要求時ページングの方式である．この方式においては，プログラムの実行時点にページ単位で実記憶装置のスペースを割り当てるので記憶域断片化の問題は解消される．しかも，良く使われるプログラム部分は常に記憶装置上に留めておくというソフトウェア・アルゴリズムを組み合せることによって，割り当てられているがほとんど使われていないスペースの問題も同時に解決され

る．これによって，プログラマに大きなアドレス空間が与えられるだけでなく，実記憶装置の有効利用については，新たなハードウェア機構とそれをたくみに使用して主記憶域の管理を行う制御プログラムに委ねることができるのである．

以上みてきたように，プログラマにより大きなアドレス空間を与え，実記憶装置の使用効率を向上させるためには，要求時ページングにもとづいた仮想記憶装置のシステムの考え方が最も妥当といえるのである．次章以降では，この要求時ページングにもとづいた仮想記憶装置の概念とそれをささえるハードウェアとソフトウェアの構造について考察することにする．

二．仮想記憶システムの概念的構造

前節では，コンピュータのハードウェアとソフトウェアの歴史的発展過程を振り返り，その中で主記憶装置に課されている2つの大きな課題を浮彫りにした．つまり，主記憶装置の断片化を極力排除して無駄のない効果的利用を行なう必要があり，かつ，アドレス空間の厳しい制約のためにプログラマが余計な負担や複雑なテクニックを使用するように強いられることから解放することが重要であることを指摘し，それぞれの具体的課題を明確にした．さらに，これら主記憶装置にかかわりを持つ課題を解決するための当然の帰結として，仮想記憶システムに到達せざるを得ないことを示唆した．

本節では，この仮想記憶装置の方法をさらに詳しく論じ，従来の記憶域管理と仮想記憶のそれとを比較し，仮想記憶装置の概念的構造を明らかにしたい．これ以降の論旨の統一をはかるため，仮想記憶装置の概念的構造の説明に入るまえに，前節でも簡単にふれたアドレス空間と実記憶装置について定義することから入りたい．

1.　アドレス空間と実記憶装置

コンピュータのプログラムは，アセンブラ言語のような記号言語または

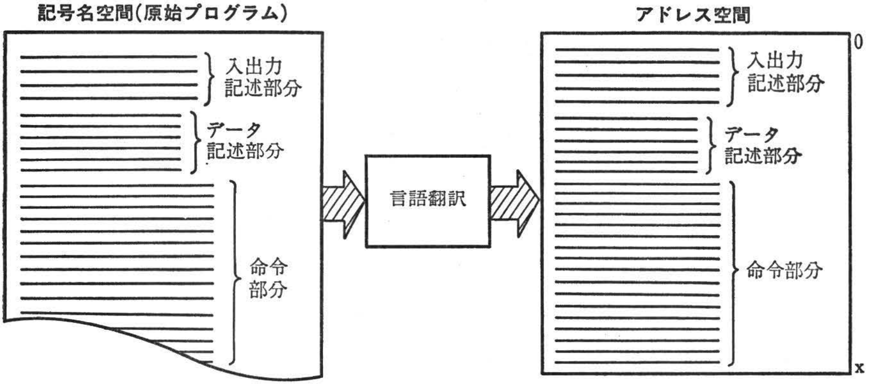


図 9-7　記号名空間とアドレス空間

FORTRAN や COBOL 言語で代表される高水準言語を使ってコーディングされる．プログラムが，どんなプログラミング言語で書かれるにしろ，そのプログラムを構成するものとしては，データの処理手順を指示する命令部分，処理のためのデータのありかを記述する部分および入出力ファイルや入出力レコードを記述する部分がある(**図 9-7** 参照)．各部分の組み合せ方は使用するプログラミング言語によって異るが，上記の各部分は一つの連続したスペースにレイアウトされ，書きあげられる．この連続したスペースは必ずしも実際のコンピュータの主記憶装置と対応している必要はなく，プログラマが頭の中で描くことのできる概念上の空間ともいうべきものである．

ところで，この概念上のスペースには一定の名前をつけて取り扱う．データ記述部分では，各々のデータの項目に名前をつけることによって他のものと区別する．入出力記述部分でも，ファイル名やレコード名などを与える．命令にも名前をつけて，ブランチ命令などで参照する．また，命令のオペランドではデータ項目名やレコード名などを指示する．つまり，プログラマが頭の中で組みたてた空間を，各種の記号名を使ってコントロールしているのである．この空間のことを一般に**記号名空間**（name space）と呼ぶ．これは，プログラマが記号命令と記号名の組合せで組立てたものであり，コンピュータで解く問題の解法が描かれているといえる．

プログラマが記号名空間に描いた結果が，いうまでもなく原始プログラムである．この原始プログラムムは言語翻訳プログラムで翻訳されて，機械語に直される．その結果，記号名は主記憶装置の実際のアドレスに対応づけられる．また，各々の命令にもアドレスが割り振られる．このように命令やデータで参照することのできるアドレスの集合体が**アドレス空間** (address space) である．図9-7の場合，0からxまでの範囲がアドレス空間である．なお，このようなアドレス空間は，別のアドレス空間と連結されてさらに大きなアドレス空間になることもできる．このような作業に使われるのが連係編集プログラムであることは今更指摘する必要もあるまい．

ところで，さきに説明した記号名空間は，プログラマが頭の中で完全に自由に描くことはできない性格のものである．アドレス空間の大きさを前提としてはじめて描くことのできる空間である．言いかえれば，記号名空間はアドレス空間に完全に制約されている．しかも，これまでのコンピュータでは，このアドレス空間が，導入されているシステムの主記憶装置の大きさを越えることは許されなかったし，実際には監視プログラムが占める記憶域などのために，アドレス空間は主記憶装置の大きさの何割りかにせばめられた(図9-8左側参照)．

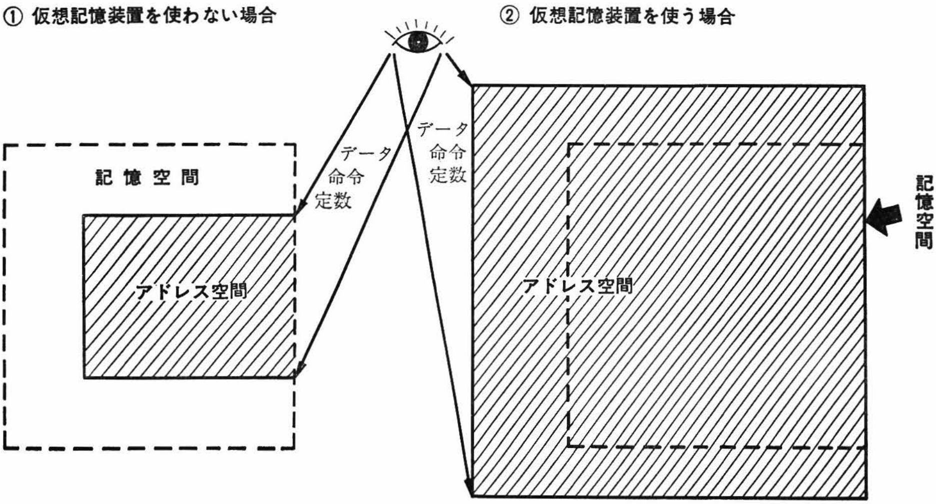


図 9-8　アドレス空間と記憶空間

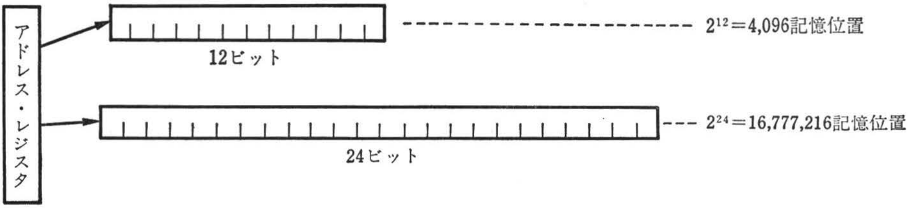


図 9-9　実記憶装置のアドレスの上限

多重プログラミングの場合だと，この記憶空間はいくつかの区画に分割されるので，アドレス空間はさらに厳しいものとなった．そこで，仮想記憶装置では，アドレス空間が記憶空間にしばられることなく，記憶空間を越えた大きさで存在することを自由にしようと試みるのである．アドレス空間が大きくなれば，それだけプログラマが描くことのできる記号名空間は拡がり，厳しい制約から解放されてプログラムを作成することが期待できることになる．

なお，図 9-8 で示している**記憶空間**(storage space)とは，単に導入されているシステムの主記憶装置の大きさを意味しているに過ぎない．

プログラマが描く記号名空間を制約するアドレス空間に対し，コンピュータに備えられている主記憶装置のことを特に**実記憶装置**（real storage）と呼ぶ．仮想記憶装置では，アドレス空間は主記憶装置にしばられるのではなく，仮想記憶装置にだけ制約される．しかも，使用者の立場からみれば，仮想記憶装置がアドレス可能な記憶域であり，従来の主記憶装置と全く同じようにみなすことのできる概念上の記憶装置である．したがって，**仮想記憶装置**（virtual storage)と明確に区別するために，従来の主記憶装置を実記憶装置と呼ぶのである．

実記憶装置をどの位まで大きくすることができるかということは，CPU にある**アドレス・レジスタ**（address register）のサイズによってきまってくる．**図 9-9** で示すように，12ビットのアドレス・レジスタは4096個の記憶位置（バイトないしはワード）まで指すことができるし，24ビットのアドレス・レジスタを持っている場合には，16,777,216の記憶位置まで指すことができる．このように論理的には，アドレス・レジスタで表現しうる最大の値がアドレス可能範囲ということになるが，実記憶装置はそれほど安価ではないので，実際には

アドレス・レジスタで示し得るアドレスの何分の1かの範囲に固定されているのである．たとえば，24ビットのアドレス・レジスタを持っているシステムでも，512Kとか1024Kあるいは2048Kといった記憶域の大きさに固定されている．しかし，論理的にはさきほど示したように16,000Kバイトまで表現することができる．したがって，この論理的サイズを仮想記憶装置の記憶域の大きさに適用してゆこうという発想がでてくるのである．これに関しては，あとで詳述する．

2.　絶対再配置と静的再配置

つぎに，プログラムが実際にコンピュータで実行される時点に焦点を合わせてみよう．さきにふれたように記号名空間は，言語翻訳プログラムによってアドレス空間に変換され，記号名がアドレスと関係づけられる．しかし，このアドレスは，必ずしもプログラムが実行される時点での実際のアドレスを意味しない．そこで，翻訳されたアドレス空間を何らかの方法で実記憶装置の実際のアドレスに結びつける必要がある．このプロセスのことを一般に**再配置**(relocation）と呼んでいるのである．

再配置は，前節最後で簡単に紹介したように，それが行なわれる時点によって，絶対再配置，静的再配置，動的再配置の3種類に分けられる．この再配置は，比較的高価な実記憶装置をいかに経済的に活用するかという点で重要な役割りを果すだけでなく，プログラマの作業をやりやすくするかどうかという面にも大きな影響を与える．というのは，すでにふれたようにオペレーティング・システムがうまく実記憶装置を管理してくれるなら，実記憶装置の不足のためプログラマが余計な仕事をしなくても済むようになるからである．そこで，絶対再配置と静的再配置の方法とその限界を明確にしたうえで，仮想記憶装置に適用される動的再配置の手法を検討してゆくことにしよう．

絶対再配置は最も単純な方法である．アセンブルまたはコンパイル時に作り出されたアドレス空間をそのまま実記憶装置上のアドレスに適用してしまうやり方である．たとえば，**図9-10**においてプログラムAがコンパイルされて出来上ったアドレス空間，つまり，0から50K番地までがそのまま実記憶装置の0

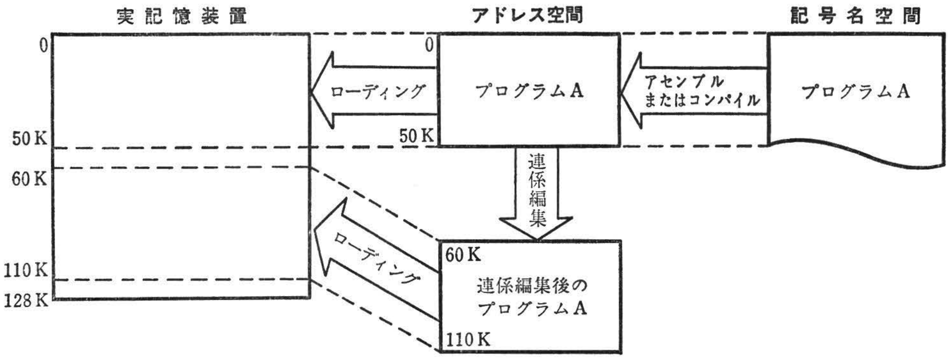


図 9-10 絶 対 再 配 置

から50K番地に対応づけられ，この場所にプログラムをローディングしてしまう．しかも，プログラムの実行中は，コンパイル時に決定されたアドレスがそのまま使われる仕組みになっている．

絶対再配置の別な方法もある．これは，コンパイルまたはアセンブルによって得られたアドレス空間を連係編集プログラムを通すことによって，別のアドレス空間に再配置するやり方である．いったん連係編集すると，それ以降は必ずその場所で実行しなければならないが，連係編集のステップをやり直すことによって異った再配置を行ない別の場所でも実行することができるようになる．ともあれ，連係編集プログラムで再配置を行うことによって，多重プログラミングにも適用できるようになる．つまり，連係編集をやり直すことによって，多重プログラミングのもとで，他の区画でも同一プログラムの実行が可能となる．

コンパイル時またはアセンブリ時にせよ，連係編集時にせよ，いずれの場合にもプログラムが実行されるさいに用いられる絶対的なアドレスが事前に定まってしまうので，これらは**絶対再配置** (absolute relocation) なのである．

絶対再配置は，連係編集プログラムを用いることによって多重プログラミングにも適用できるプログラムを作り出すことはできるが，どの区画にどのプログラムをローディングすべきかという計画を立てる必要があるし，同一プログラムを異った区画でも実行できるようにするためには，ローディング・ポイン

トを異にした2つのロード・モジュールを用意するといったわずらわしさが残る．何よりも問題となるのは，ある区画が完全に空いていても，運悪くその区画にローディングできるようなロード・モジュールに仕立てあげていないときは，その区画は遊んでしまうという事態が生じてしまう．プログラムの実行はきわめてダイナミックであるだけに，事前に各々の区画の利用状況を予想して，ジョブのスケジュールを行なうことは至難なことであるだけに，このような記憶域の大幅な断片化は，絶対再配置を採用した場合には回避できないことである．

この問題は，静的再配置を用いると解決することはできる．静的再配置では，実記憶装置のどの区画で実行するかは，プログラムが実行される直前，つまりプログラムのローディング時に決定されるからである．

静的再配置 (static relocation) の様子を示したのが**図 9-11** である．コンパイルや連係編集が終ったあとも，絶対再配置の場合とはちがって，実記憶装置の記憶位置との絶対的結びつきはない．アドレス空間は単に相対的なアドレス関係を示しているだけである．ところが，プログラムをある区画にローディングするさいに，実記憶装置の実際のアドレスがきめられてゆく．それゆえ，図 9-11で示しているように，プログラムAは第1回目の実行のときには実記憶装置の90Kから160Kバイトのところにロードされ実行されるが，第2回目には210

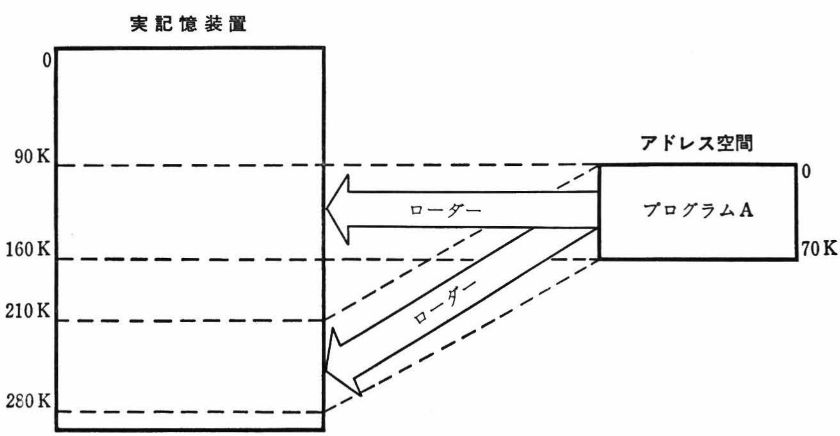


図 9-11 静的再配置

Kから280Kバイトの場所にロードされ実行されるのである．

このような静的再配置の方法により，多重プログラミングをより柔軟に行なうことはできるが，いったんある区画にロードされてしまうと，もはやその場所から動かすことはできない．その結果，ロールアウト／ロールインの機能を用いる場合でも，ロールアウトされたプログラムは，再び実記憶装置に戻ってくるときは，前の区画にしかロールインできないのである．このような性格があることから，静的といわれるのである．

この静的という性質と，プログラムは連続した場所でしか実行できないという性質が組み合わさると，記憶域の断片化に関して始末におえない状況を呈する．これは第1節でも指摘したことであるが，断片化の記憶域の総容量が，つぎに実行すべきプログラムを収容するに十分であっても，連続した区域でないために，そのプログラムの実行はむなしく待されてしまうのである．この問題を解決するための1つの方法に，実記憶装置中のプログラムを詰め込むやり方，すなわち，記憶域コンパクションがある．しかし，静的再配置では，プログラムがいったんロードされると，そのプログラムを実行中に他の場所に動かすことはできない．それが全く不可能でないとしても，動かした分だけプログラム中のアドレス調整を行なわなければならない．この処理をオペレーティング・システムで面倒みるという方法も考えられるが，非常に時間がかかる仕事であり，またプログラミング上に厳しい約束事をつけ加えなければならないことから，実用化はなかなか困難である．そこで，何らかのハードウェア機構とからみ合わせて解決される方法がとられる．それが**動的再配置**(dynamic relocation)である．

3. 動的再配置による方法

動的再配置のハードウェアについては，これまで数多くの研究がなされ，いろいろな機構が考えられてきた．そこで，この項では，

1）再配置レジスタによる方法

2）セグメンテーションによる方法

3）ページングによる方法

をとりあげて解説し，仮想記憶装置の考え方を明らかにしてゆきたい．

ところで，静的再配置ではプログラムがローディングされるときに絶対アドレスが決定されたが，動的再配置ではプログラムがローディングされたあとも相対アドレスのままである．これらのアドレスが実際の物理的アドレス，つまり絶対アドレスに変換されるのは，プログラムの実行中である．換言するならば，プログラムの実行中に命令やデータが参照されるつど，自動的に相対的アドレスが絶対的アドレスに変換される仕組みである．この種の変換を**動的再配置**と呼び，**動的アドレス変換機構**という特別なハードウェア機構でアドレス変換が行なわれる．

1）再配置レジスタによる方法

再配置レジスタによる方法は，記憶域コンパクションに適用される動的再配置の方法として好都合なものである．

再配置レジスタ（relocation register）は再配置のための特殊なレジスタで，このレジスタには処理プログラムが記憶装置上にローディングされたさいの起点がポインタとして貯えられる．そして，アドレス空間上の相対的アドレスが参照されるつど，アドレス算出のためのプロセスが行なわれる（**図 9-12** 参照）．すなわち，アドレス空間上の相対アドレス，つまり起点を0とした場合の変位に，再配置レジスタの中味である該当プログラムの実記憶装置上の起点が加算され，絶対アドレスが決定されるのである．

この方法により，実行の途中でプログラムを実記憶装置の中で動かしても，それを反映して再配置レジスタの内容だけ更新すれば，全てがうまくゆくはずである．したがって，静的再配置のところで問題にした記憶域の断片化の対策として採用される記憶域コンパクションが比較的容易に実現できる．しかし，この方式では，プログラムが実記憶装置上の連続した区域にローディングされなければならない．この結果，アドレス空間が実記憶装置の大きさより小さくなければならないという制約から逃れることはできない．したがって，仮想記憶装置の大前提となるプログラムに対してより大きなアドレス空間を与えるという課題は，再配置レジスタを用いた動的再配置では解決できないといえ

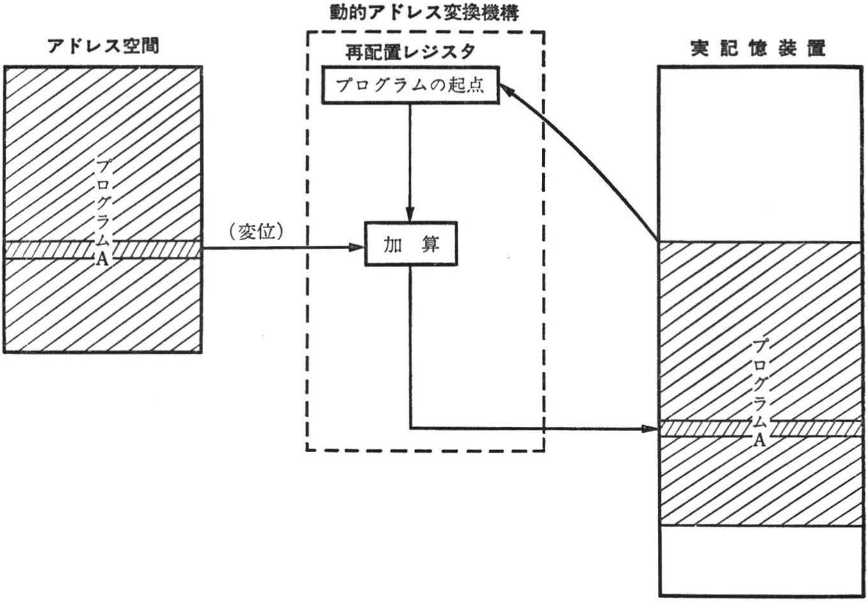


図 9-12　再配置レジスタによる動的再配置

る．

2）　セグメンテーションによる方法

セグメンテーションによる方法は，第１節で簡単に説明したスキャター・ローディングの一手法である．

図 9-13の左側をみていただきたい．目下，プログラムの実行を待っているのがプログラムＤである．静的再配置のシステムだと，プログラムＤは，実記憶装置の連続した50Ｋの区域が空かないことにはローディングできない．空いている区域の合計が50Ｋ以上あったとしても，それは連続していないので実行できない．したがって，50Ｋは無駄になってしまう．この無駄を排除するためには，どうすればよいだろう．その１つの解決法が，前述の再配置レジスタによる方法である．もう１つは，セグメンテーションによるスキャター・ローディングによる方法だ．

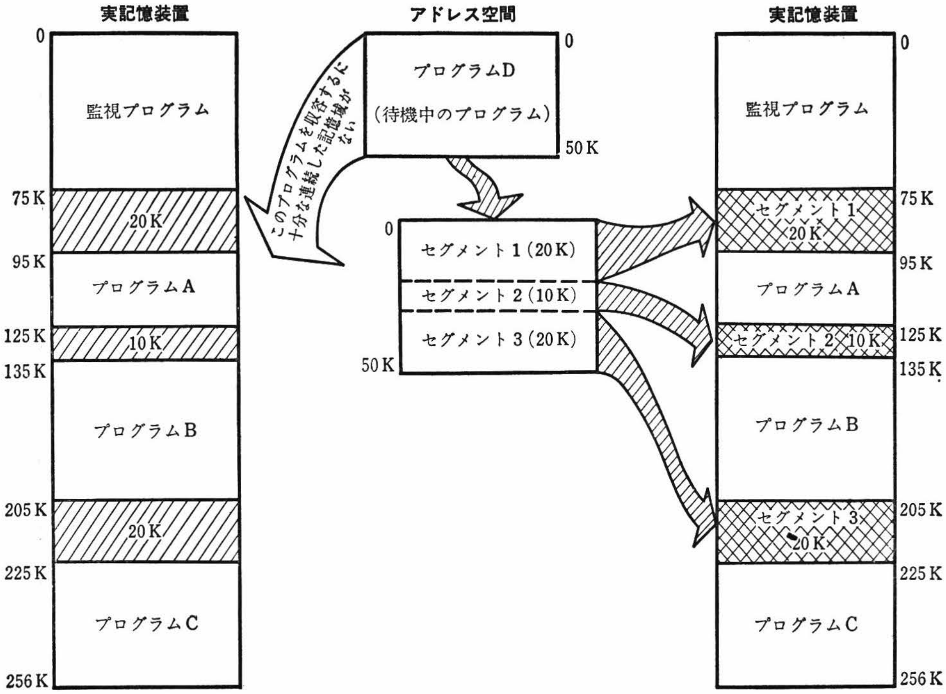


図 9-13　セグメンテーションによるスキャター・ローディング

このために，プログラムＤのアドレス空間を**セグメント**(segment) と呼ぶ３つの部分に分割する．図9-13に示すようにセグメント１(20K)，セグメント２(10K)，セグメント３ (20K) である．このセグメント単位で，各々の空きの区域にローディングする手法を採用すれば，断片化によって遊んでいる実記憶装置が有効に使えるようになるのである．

このようなセグメント化は，オペレーティング・システムが行なうべきであって，プログラマに負担をかけるものであっては意味がない．ともあれ，各セグメントのアドレスは０から始まる連続した空間である．セグメント化されたプログラムもやはり相対アドレスを持つ．**図 9-14** で示しているように，セグメンテーションの場合の相対アドレスは，セグメント番号と変位で構成される．**セグメント番号** (segment number) は，アドレス空間の何番目に位置するセグメ

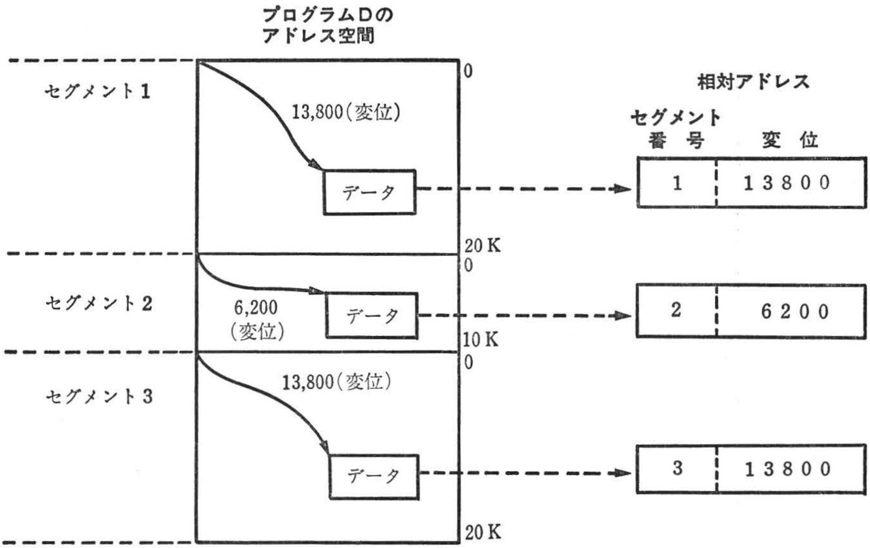


図 9-14 セグメンテーションと相対アドレス

ントであるかを示す．つまり，セグメント化されたプログラムの論理的順序を示している．一方，**変位**（displacement）はセグメント内での相対的位置（つまり，セグメントの最初を0とした場合のへだたり）を表わす．なお，セグメント番号にしろ，変位にしろ，どれだけの大きさまでの表現が許されるかは，それぞれのために設けられたビットの数できまる．たとえば，セグメント番号に8ビットあてられていると0から256番までのセグメントが表現できる．一方，変位として16ビット設けられていると，0から65,536までのアドレスが表現できる．このように，相対アドレスをどのように分割するかによって，セグメントの最大数と最大の大きさが決まるのである．ともあれ，セグメント化したプログラムでは，命令やデータを示すアドレスは，2つの要素，セグメント番号とセグメント内での相対的位置を示す変位とで構成される．この相対アドレスは，セグメント化されたプログラムのアドレス空間に対応しているのである．

セグメント化されたプログラムを実行するためには，プログラムの実行中に命令やデータが参照されるつど，アドレス変換を行なわなければならない．こ

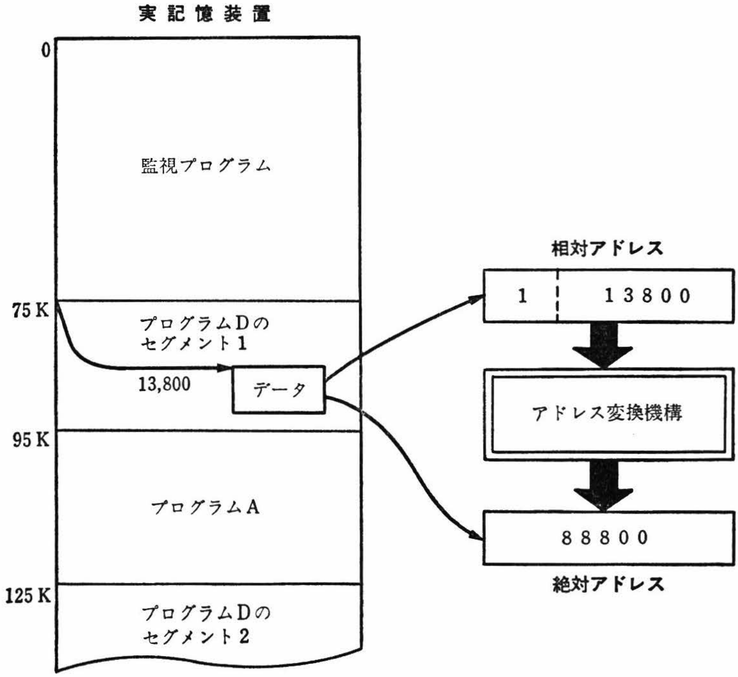


図 9-15　相対アドレスから絶対アドレスへの変換

のためには，やはり特別なハードウェア機構，つまり動的アドレス変換機構が必要である（**図 9-15** 参照）．相対アドレスは，セグメント番号と変位から構成されているので，セグメント番号を手がかりに該当セグメントが収容されている実記憶装置の起点のアドレスを見つけだし，このアドレスに変位を加算することによって絶対アドレスを得る仕組みが用意されていればよいことになる．この様子について，少し詳しく検討してみよう．

図 9-16 の右側では，先述したプログラムDがセグメント化されて，スキャター・ローディングされた状態を示している．オペーティング・システムがプログラムDのセグメントを実記憶装置にローディングするさいには，アドレス変換は全く行なわれていない．したがって，プログラムDの各セグメントは依然として相対アドレスのままである．ところで，CPU が命令を実行するために

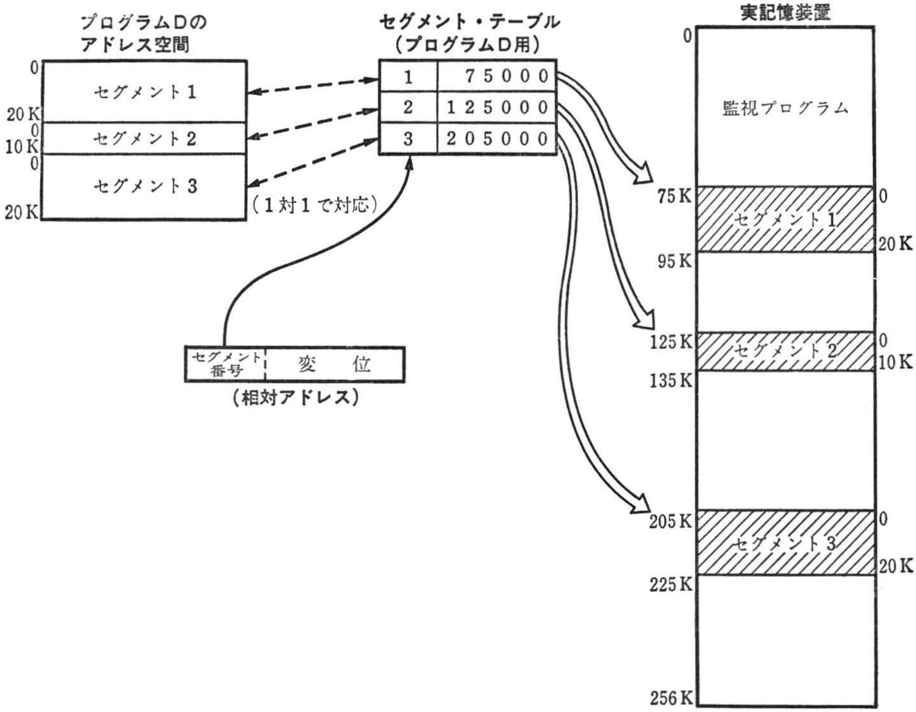


図 9-16　セグメント・テーブルによる相対アドレスと絶対アドレスの関係づけ

は絶対アドレスを使って，実記憶装置の特定命令やデータを参照しなければならない．そこで，CPU が命令を実行する前にプログラムDの相対アドレスを絶対アドレスに変換する必要が生じる．プログラムの実行中，相対アドレスを各々の命令を実行する直前に絶対アドレスに変換することを，**動的アドレス変換**(dynamic address translation) と呼ぶのである．

では，実行中にどのようにして変換するのであろうか．このためには，いうまでもなく新たなハードウェア機構を必要とするし，オペレーティング・システムでは，変換中にセグメント番号にもとづいて参照する特別なテーブルを用意する必要がある．図 9-16 の中味で示している**セグメント・テーブル**(segment table) がこれに当る．このテーブルは，実記憶装置の特定の場所に収容してお

く．セグメント・テーブルの各記入項目では，各セグメントに対応する実記憶装置内のセグメントの起点アドレスを示す．図9-16において，サグメント1は実記憶装置内の75,000番地から，セグメント2は125,000番地から，セグメント3は205,000番地から始っていることをセグメント・テーブルで示している．このセグメント・テーブルの該当項目を相対アドレスのセグメント番号でつきとめ，起点のアドレスに変位を加算すればよいのである．このようなセグメント・テーブルを実行中のプログラムごとに作成すればよいわけである（もちろんこの仕事はオペレーティング・システムがプログラムをローディングするつど行なうのである）．なお，実行中のプログラムのためのテーブルを作りあげる操作を一般に**マッピング**（mapping）と呼ぶ．つまり，セグメント・テーブルは実記憶装置内のセグメントの開始点を正確にマップしているのである．

ところで，このセグメント・テーブルの存在場所を迅速につかむためには，**セグメント・テーブル起点レジスタ**（segment table origin register）と呼ばれる特別な制御レジスタが必要である（**図9-17**参照）．このレジスタは，いつでも目下実行中のプログラムのためのセグメント・テーブルの起点アドレスをさすようになっている．このセグメント・テーブル起点レジスタとセグメント・テーブルを用いて動的アドレス変換を行ってくれるのであるが，その様子は，図9-17で示している通りである．つまり，つぎの3つのステップでアドレス変換が達成されるのである：

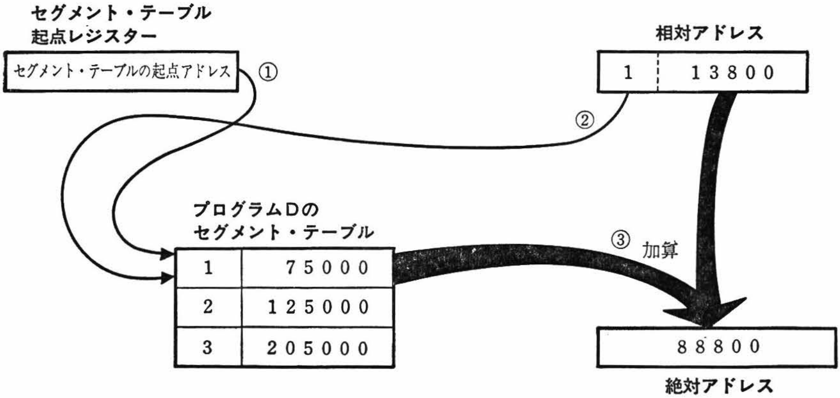


図 9-17　セグメント・テーブルによる動的アドレス変換

① セグメント・テーブル起点レジスタが，プログラムD用のセグメント・テーブルの起点をさす．

② 相対アドレスのセグメント番号を使って，セグメント・テーブルを捜す．この例では，セグメント番号はテーブルの最初の項目を示すようになっている．

③ セグメント・テーブルの項目が示している記憶位置は，実記憶装置内のセグメントの起点を示している．したがって，このアドレスに相対アドレスの変位を加算することによって，絶対アドレスが得られるのである．この例の場合，セグメント1の起点（75,000）に変位（13,800）を加えて，絶対アドレス88,800を得ている．

このようにセグメンテーションの場合のアドレス変換は，セグメント・テーブル起点レジスタとセグメント・テーブルを使って自動的に行なわれるのである．なお，プログラムの実行中にあるセグメントを別の記憶域に移動しても問題は生じない．セグメント・テーブル内の該当するセグメントの対応項目の起点アドレスを書きかえるだけで，プログラムの実行は再開できるからだ．

このようなセグメンテーションの方式では，コンパイラがプログラムをモジュール化しやすいように設計されている場合には非常に有効であるし，静的再配置のシステムより，ずっと断片化による無駄は減少する．セグメントは，プログラムの大きさより小さいので，それだけ断片化も少なくなろう．それでも，まだ断片化が存在していることは事実である．そこで，このセグメンテーションに多少変更を加えることによって，記憶域の断片化をほとんど排除してしまう方式，つまりページングによる方式が出現するのである．

3） ページングによる方式

プログラムを分割してローディングするということに関して再考してみよう．普通，仮想記憶装置は実記憶装置よりもはるかに容量が大きい．プログラマはこの仮想記憶装置上のアドレス空間を想定してプログラムを書いてゆけばよいわけであるが，完成したプログラムが実行されるときは実記憶装置に入っ

ていなければならない．しかし，プログラムの実行にさいしては必ずしもプログラム全体が実記憶装置上にローディングされている必要はない．最低限目下の実行に関係する部分だけが実記憶装置上にあればよい．したがって，プログラムを何らかの形で分割し，少しずつ実記憶装置上に持ち込んで実行するという方法が考えられる．その1つが前述したセグメンテーションによる方法である．セグメンテーションの手法を採用することにより，記憶域の断片化をかなり排除することができるし，仮想記憶装置を想定することによってプログラマのアドレス空間を拡大することもできよう．とはいうものの，セグメントのサイズによっては，断片化が十分に解決できない面も残る．そこでページ単位に分割する方法がもう1つの方法として考えられるのである．セグメント単位でローディングすることをセグメンテーションと呼んだのに対し，ページ単位でローディングすることを**ページング**（paging）と呼ぶのである．

ページングの方式では，仮想記憶上のアドレス空間をある大きさのページに分割する．ページは，固定した大きさで，1024バイトとか2048バイトあるいは4096バイトの単位が適用される．実記憶装置もこれに合わせて同じサイズの**ページ枠**（page frame）に分割される．ページ枠とは，1ページ分を収容する実記憶装置上のブロックである．プログラムのローディングは，ページ単位に実記憶装置上のページ枠に行なわれるのである．ところで，ページの大きさは断片化を排除するという観点だけからすれば，小さければ小さい程よいことになる．**図9-18**で示しているように，プログラムを構成するページの中で最後のページには，**ページ内断片化**（intra-page fragmentation）が生じる．したがってページの大きさが小さいほど，この無駄な量は少なくなる．しかし，ページをやみくもに小さくするということも，ページングのメカニズムを考慮に入れると賢明ではない．というのは，ページ単位の出し入れが多くなり，それだけ全体の処理効率に影響するし，また後述するページ・テーブルが占める場所が大きくなり，実記憶装置を余計に食うなどの欠点が生じるからである．いろいろの実験の結果，今では先述したような1024バイト，2048バイト，あるいは4096バイトなどのサイズが妥当であるとみなされるようになっている．

ともあれ，プログラムはページ単位で実記憶装置に持ち込まれるわけである

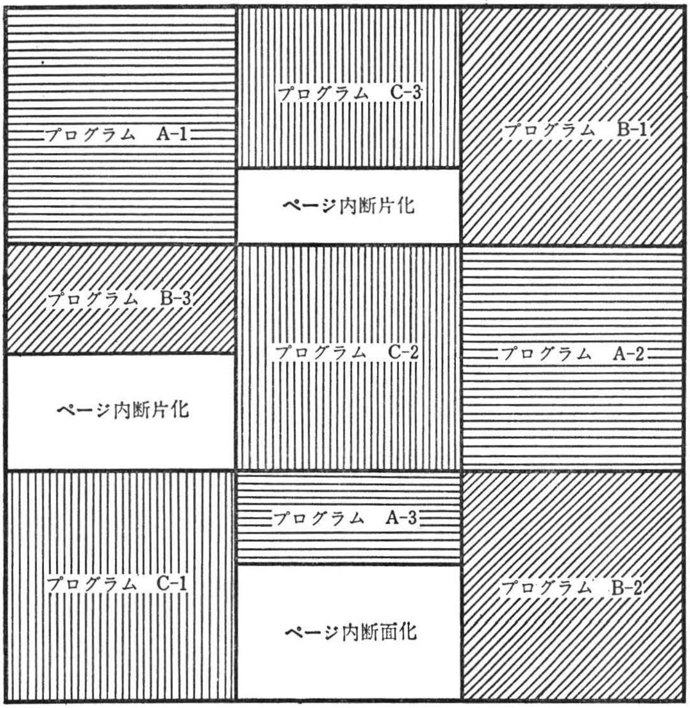


図 9-18 ページングと断片化

が，限られた実記憶装置を有効に使うために，他のプログラムやページにページ枠を融通し合う必要がある．その結果，そのプログラムの実行中各ページが実記憶装置上のページ枠の中に居座わりつづけるというわけにはいかない．そこで，実記憶装置と補助記憶装置上の**外部ページ記憶装置**（external page storage）との間を出たり入ったりする．このことを称してページングと呼んでいるが，ページアウトしたページをページインする場合，同じページ枠に帰ってくる必要がないようにしないことには，ページ単位に分割した意味が半減してしまう（**図 9-19** 参照）．そこで，セグメンテーションの場合と同様に動的再配置のメカニズムでコントロールするようにしておく．ページング・システムの動的再配置のメカニズムは，セグメンテーションの場合のそれと本質的には変らないので，簡単に説明することにする．

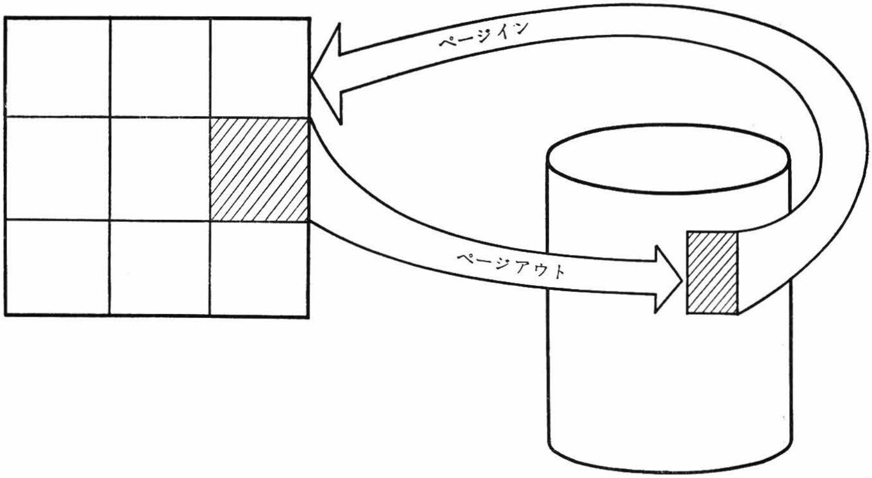


図 9-19　ページアウトとページイン

システムは，仮想記憶域の各ページがどのような状態にあるか知っている必要がある．実記憶装置上のどのページ枠にあるか，あるいは実記憶装置上にはなく外部ページ記憶装置上にあるかを把握しておく必要がある．このために，**ページ・テーブル**（page table）と呼ばれる制御ブロックを用意し，このテーブルで仮想記憶装置をマッピングしている．このテーブルはセグメンテーションの場合のセグメント・テーブルに相当するものであり，仮想記憶装置上の各ページに対応した記入項目を持っている（**図 9-20** 参照)．それぞれの項目には，対応するページが実記憶装置にある場合にはそのページ枠の最初のアドレスが示され，ない場合には意味のない記号が入っている．このページ・テーブルを使用して，命令が実行される直前に動的アドレス変換を行なうのである．ページング方式の場合，仮想記憶装置上でのアドレスは，ページ番号とページ内での変位の2つの要素で構成されている（**図 9-21** 参照)．なお，仮想記憶装置上での記憶位置を示すアドレスのことを一般に**仮想アドレス**（virtual address）と呼び，実記憶装置上の実際のアドレスのことを**実アドレス**（actual address）と呼んでいる．

セグメンテーション方式の場合だと，セグメント・テーブルの場所を示すた

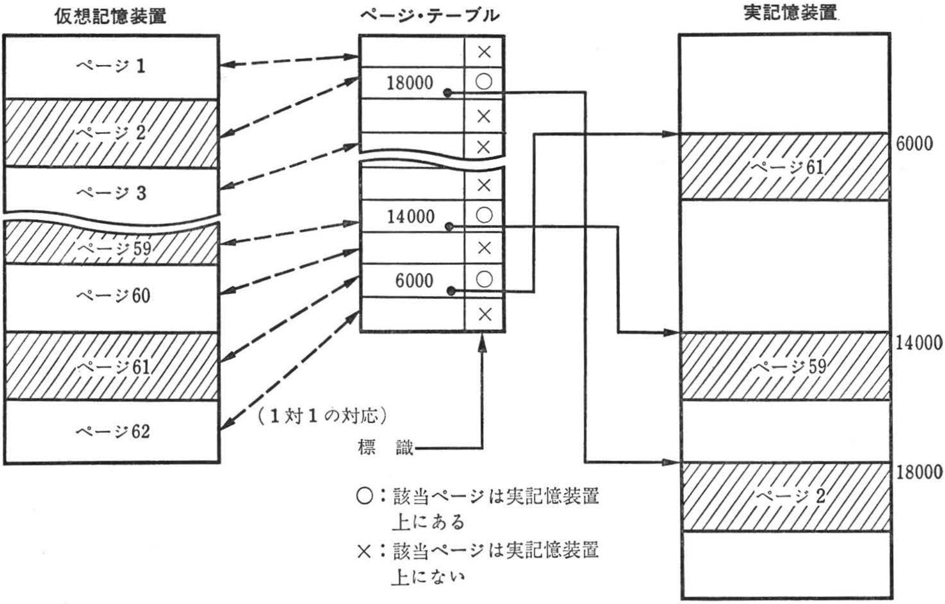


図 9-20　仮想アドレスと実アドレスの関係づけ

めにセグメント・テーブル起点レジスタが用意されていたが，ページング方式の場合には，ページ・テーブルの起点を示すためのハードウェア機構としてやはり特別なレジスタが適用される(**図 9-22** 参照). 仮想アドレスを実アドレスに変換する必要性が生じると，まず制御レジスタをもとにページ・テーブルがつきとめられる．そのあとで，仮想アドレスのページ番号にもとづいてページ・テーブルの何番目の記入項目を参照すべきかがきまる．図9-22ではページ番号が01であることから，2番目の記入項目を使う．必要なデータが入っているページは，実記憶装置の18,000番地からのページ枠にあることが判明する．したがって，この18,000に仮想アドレスの変位1032を加算することによって実アド

ページ番号	変　位

図 9-21　仮想アドレスの構造

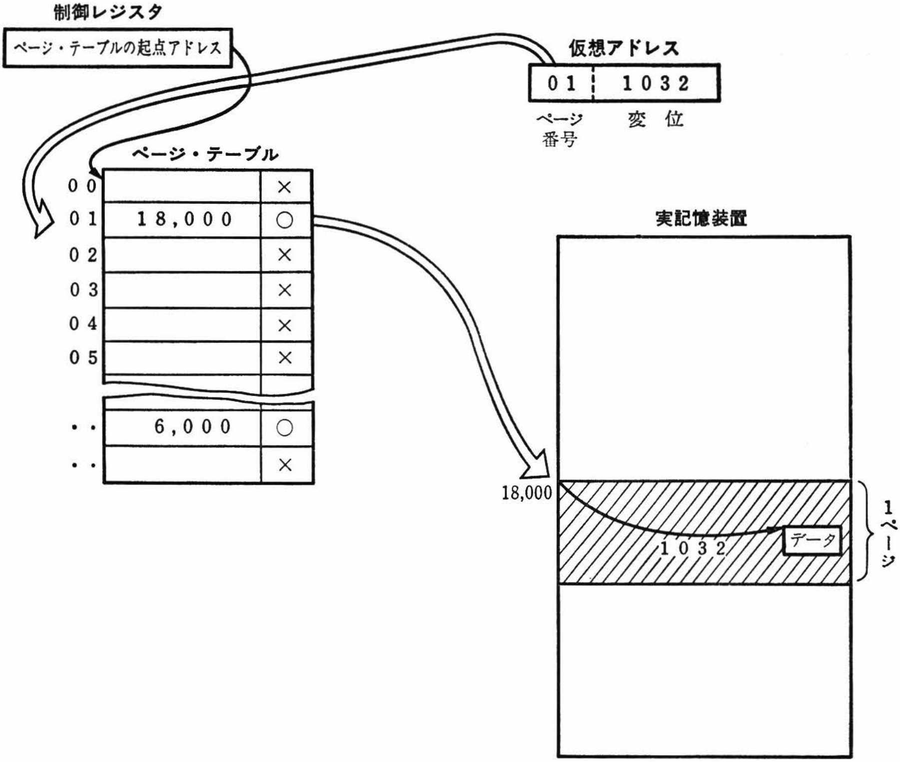


図 9-22　ページ・テーブルによる動的アドレス変換

レス19,032が得られることになる．もし，求めるページが実記憶装置に存在しないことを，ページ・テーブルの記入項目が示している場合だと，ページ不在の割込みを起してページ監視プログラムを呼び，空きのページ枠ないしは空きのページ枠がないときには，使用頻度の低いページをページアウトしたうえで，該当ページのページインを行なう．そのあとで，前述の手順を踏んで動的アドレス変換を達成するのである．

以上みてきたように，ページング方式もセグメンテーションによる方法と本質的には異なるところがない．異なるのはページング方式の場合，実記憶装置に呼び込まれる単位がページの大きさに固定されているが，一般にセグメンテーションの場合には，セグメントの大きさは固定されていないのが普通であ

る．セグメントのサイズを固定しない理由の１つは，セグメントを論理的な意味のあるまとまりのあるプログラム・モジュールにしようというところにある．プログラミングの能率向上，保守の容易化などの観点から，プログラムのモジュール化が注目されるようになり，ますますこの傾向は強まっているが，これにハードウェア面からも適合させようという目的で出現してきたのがセグメンテーションであり，必然的にセグメントの大きさを固定しない方向が目指されたのであろう．ともあれ，セグメンテーションは，ページングと異なり，プログラムの出し入れの最小単位が論理的なモジュール単位となるので，プログラムの実行中でのセグメントアウトとセグメントインの頻度は少なくなると予想される．したがって，ページング方式にくらべ，プログラム部分の出し入れという面から見ると実行効率は上るであろう．しかし，実記憶装置のどこへセグメントを配置するかという面での管理はむずかしくなり，サイズが固定されたページング方式とは異なり，どの場所にでも持ち込めるという保障はない．つまり，必要なセグメントの大きさが，空いている区域に十分入りきるかどうかが問題となる．その結果，記憶域の断片化の問題が依然として残されてしまう．そこで，ある程度妥協してセグメントのサイズを何種類かの大きさにきめておき，セグメントの断片化をある程度防止しているようなシステムもある．

ともあれ，セグメンテーション方式もページング方式もいくつかの難点をもっているのは事実である．したがって，別のアプローチが出現する．その１つが，セグメントとページを組合せた方法である．つまり，セグメントとページという２つのレベルのテーブルを設けて，動的再配置を行なうシステムである．

4. セグメンテーション・ページング方式による仮想記憶システム

セグメンテーション・ページング方式では，概念上の記憶装置である仮想記憶装置をセグメントとページの２つの階層に分割して取り扱う．プログラマは，この仮想記憶装置が提供するアドレス空間を念頭にプログラミングすればよい．ところで，仮想記憶装置上をセグメントあるいはページに分割して処理

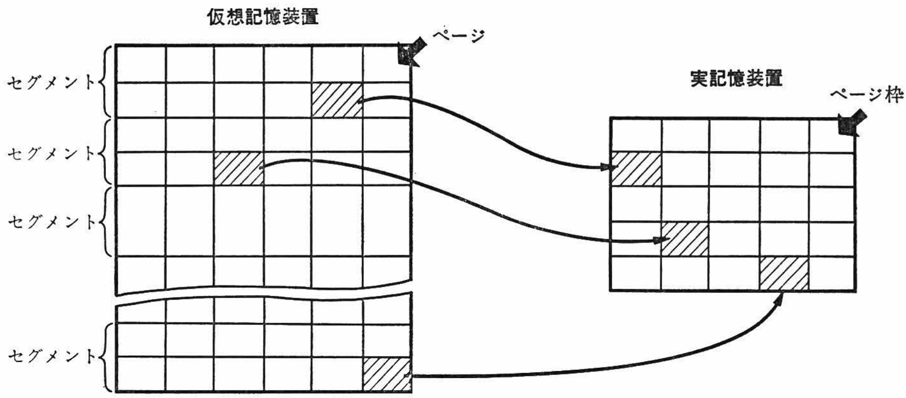


図 9-23　セグメントとページの関係

するのは，ハードウェアと制御プログラムの機能であって，何らプログラマを拘束するものではない．ともあれ，セグメンテーション・ページング方式では，仮想記憶装置上をいくつかのセグメントに分割し，そのセグメントはさらにページ単位に分割して取り扱われる(**図 9-23** 参照)．図でも分るようにセグメントは，複数個のページから構成される仮想記憶装置上の連続した区域であり，ページは実記憶装置との間で転送される命令やデータのブロックである．一般には，セグメントもページも固定した大きさである．セグメントは，64Kバイトの大きさが普通である．一方，ページはこれまで再三指摘したように2Kバイトあるいは4Kバイトがよく用いられている．したがって，1セグメントは16ページないしは32ページから構成される．

セグメンテーション・ページング方式の動的アドレス変換の概念的構造を説明する前に，仮想記憶装置と**外部ページ記憶装置** (external page storage ; 略して **EPS**) の関係について簡単にふれておこう．仮想記憶装置は，その名が示している通り実在しない記憶装置である．プログラマにより大きなアドレス空間を提供するための架空の記憶装置である．プログラマはこの概念上の記憶装置を相手にプログラムを書く．その結果のプログラムは，仮想記憶装置を想定して書きあげたプログラムであるから，実記憶装置に一度に収容できるという

保証は全くない．したがって，実行中のプログラムを構成するページのいくつかは，別の場所に一時的に保管する必要性が生じる．このために使われる補助記憶装置上の区画のことを**外部ページ記憶装置**と呼んでいる．外部ページ記憶装置は，**図 9-24** に示しているように**スロット**（slot）と呼ばれるページが格納される部分から構成されている．したがって，ページアウトとページインの操作は，外部ページ記憶装置上のストロットと，実記憶装置上のページ枠との間で行なわれるのである．なお，仮想記憶装置は実在しないものの，プログラムのローディング自体は仮想記憶装置を相手に行なわれるのである．つまり，仮想記憶装置を反映した一定の制御テーブルにもとづいて，プログラム・ライブラリーから該当プログラムを実記憶装置上にローディングしてゆく．このローデング過程においても，実記憶装置上に空きのページ枠がない場合には直ちに外部ページ記憶装置にページアウトされてしまうのである．プログラムのローディングおよびページアウト，ページインの具体的仕組みに関しては，次節以

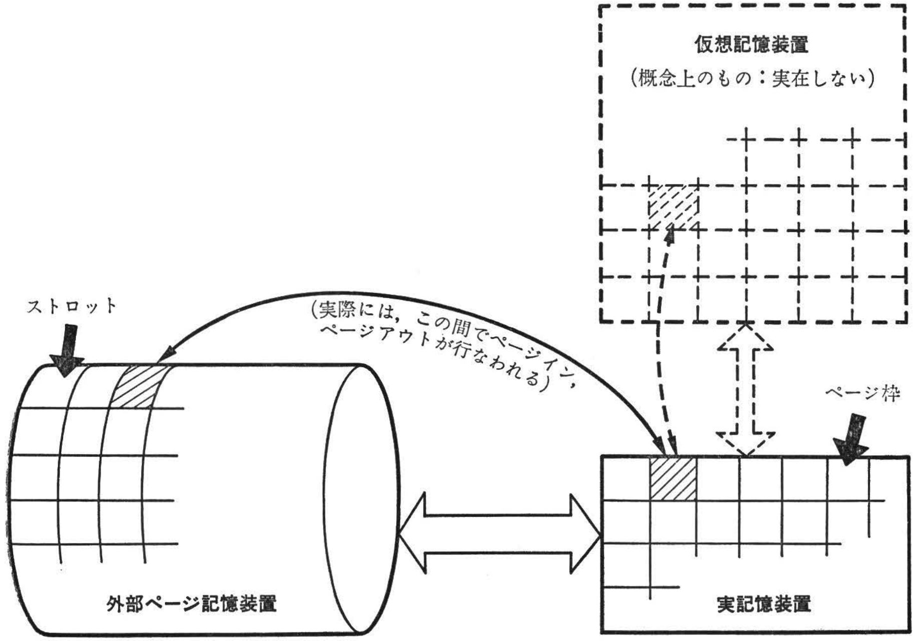


図 9-24　仮想記憶装置と外部ページ記憶装置

降で詳述する．ここでは，仮想記憶装置，外部ページ記憶装置および実記憶装置の相互関係を知るだけで十分である．

1）動的アドレス変換の方法

セグメンテーション・ページング方式の動的アドレス変換に話しをもどそう．プログラムが仮想記憶装置を相手にローディングされた状態の命令やデータのアドレスはやはり相対的なアドレスである．換言すれば，ページ単位で実記憶装置に存在するプログラムが用いているアドレスは，**実アドレス**（actual address）にはほど遠い仮想アドレスである．この仮想アドレスの形式は，**図9-25**で示しているように，セグメント番号，ページ番号および変位から構成される．セグメント番号のために必要となるビット数は，仮想記憶装置を何個のセグメントに分割するかで決定されよう．ページ番号に要するビット数は，1セグメントを何個のページで構成するかにかかっている．1セグメントが16ページで構成される場合には4ビット，32ページで構成される場合には5ビット必要となろう．変位に要するビット数は，1ページが何バイトに固定されているかで定まる．つまり，2048バイトの場合だと11ビット，4096バイトだと12ビット必要になろう．これらセグメント番号，ページ番号および変位に関しては，何らプログラマが考慮を払う必要はない．仮想記憶装置上に静的再配置の方法でプログラムがローディングされたさいに用いられている仮想アドレスを適当なビット数で区切ることによって，セグメント番号，ページ番号および変位が得られるからである．

このような仮想アドレスを実アドレスに変換するために，セグメント・テーブルとページ・テーブルが参照される（**図9-26**参照）．1つの仮想記憶装置に対

セグメント番号	ページ番号	変　　位

図 9-25　仮想アドレスの形式

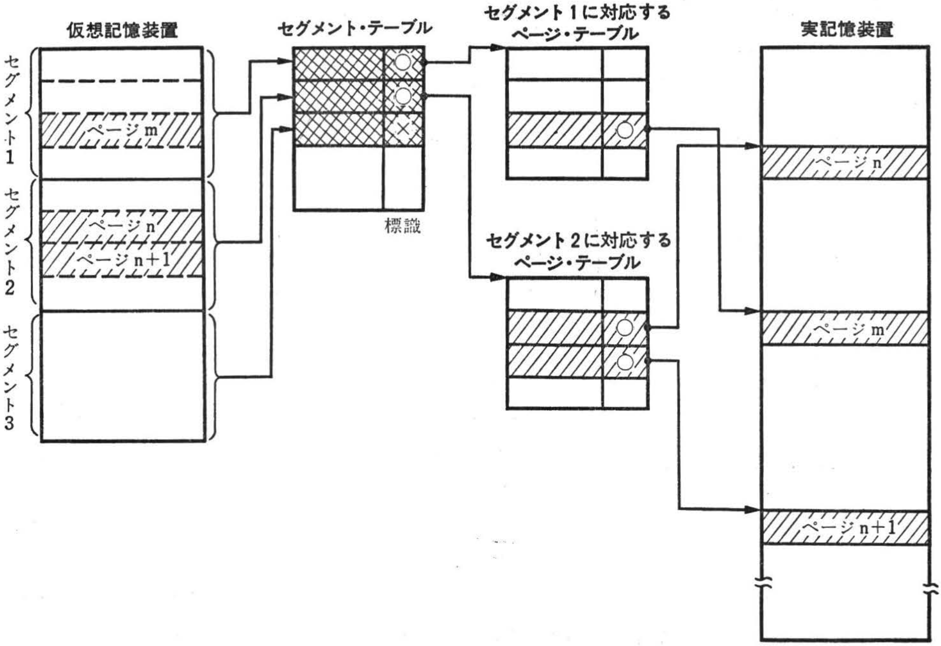


図 9-26 セグメント・テーブルとページ・テーブルによる仮想アドレスと実アドレスの関係づけ

応して1個のセグメント・テーブルが作成される．セグメント・テーブルの記入項目は，仮想記憶装置上の各セグメントごとに1個ずつ作られる．さらに，この記入項目に対応してそのセグメント内の各ページの所在を示すページ・テーブルが構成される．このページ・テーブルはさきのページング方式のページ・テーブルとほとんど同じであるが，セグメントごとに作られる点が異なる．ただし，図9-26で示しているように仮想記憶装置上のセグメントが全く使用されていない場合には，セグメント・テーブルの該当記入項目の標識欄にその旨のフラグをつける（図では便宜上×印で示している）ことによって，ページ・テーブルを省略する．そのセグメントが目下使用されていないので，ページ・テーブルが不在でも何ら支障はないのである．ともあれ，セグメント・テーブルの記入項目では，そのセグメントに対応するページ・テーブルのアドレスを

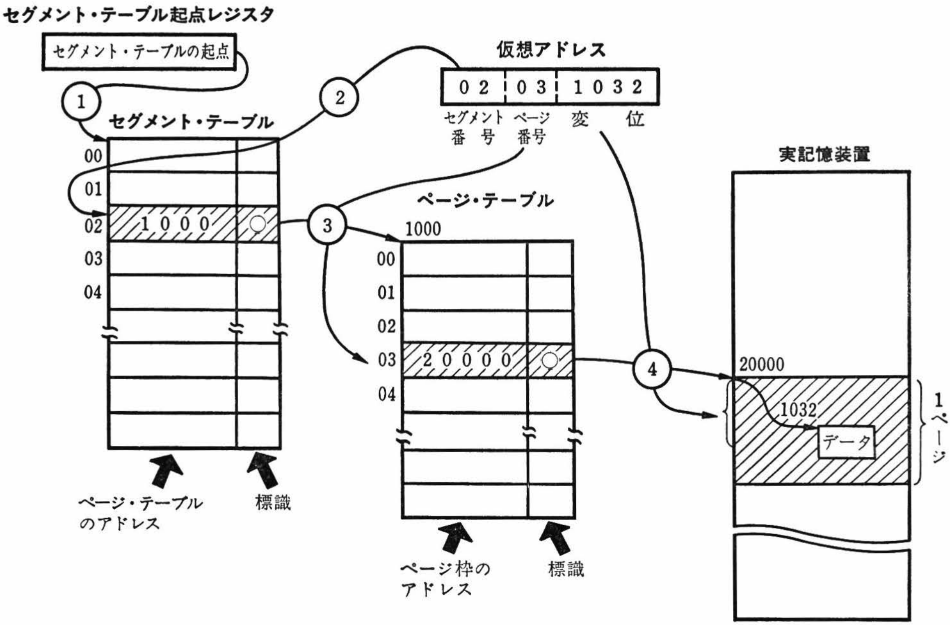


図 9-27　セグメント・テーブルとページ・テーブルによる動的アドレス変換

指す．一方，ページ・テーブルの各記入項目では，該当ページが収容されている実記憶装置上のページ枠のアドレスを示す仕組みになっている．もし，該当ページがページアウトされて実記憶装置上に存在しないときは，やはりその旨のフラグが標識欄につけられる（図では○印によって，ページが実記憶装置上に存在していることを示している）．

さて，命令が実行されるさいに，データや命令の仮想アドレスはどのようにして実アドレスに変換されるのであろうか．この様子を示したのが，**図 9-27**と**図 9-28**である．まず，図9-27で全体の流れを説明しよう．セグメンテーションやページングの方法の場合と同様，セグメント・テーブルの起点のアドレスを示す特別な制御レジスタが CPU の中に用意される．これを**セグメント・テーブル起点レジスタ**（segment table origin register；略して**STOR**）と呼ぶ．これによって，何時でもセグメント・テーブルの最初のアドレスが表示されている．なお，言うまでもないことであるが，セグメント・テーブルもページ・テ

ールも，制御プログラムが実記憶装置上に作りあげるものである．ともあれ，仮想アドレスの変換にさいしては，まずこのレジスタが参照され，セグメント・テーブルの起点がつきとめられる（図9-27の①)．さらに仮想アドレスのセグメント番号を用いて，セグメント・テーブル上の該当項入項目が選択される（図 9-27 の②)．記入項目が選び出されると，ページ・テーブルのアドレスが得られる．このアドレスと仮想アドレスのページ番号を用いて，該当ページの記入項目の場所がつかまる（図 9-27 の③)．該当ページが実記憶装置上のページ枠にあると，標識欄にその旨表示されているので，ページ枠のアドレスに仮想アドレス上の変位を加えて，実アドレスを算出するのである（図 9-27 の④)．標識欄にページ不在の標識が立っている場合だと，プログラム割込みが生じそのあとは**ページ監視プログラム**（paging supervisor）が面倒みる．つまり，使用可能なページ枠を探し出し，該当ページを外部ページ記憶装置からページインすると同時に，そのページ枠のアドレスをページ・テーブルの該当欄に挿入し，標識の部分をページが存在する旨変更する．そのあとで，前述と同じ手順によって，動的アドレス変換を行なうのである．

図 9-28で仮想アドレスから実アドレスに変換する場合の具体例を示している．起点レジスタの28,000によってセグメント・テーブルのアドレスがつきとめられ，仮想アドレスのセグメント番号をこれに加えることによって，セグメント・テーブルの該当記入項目が得られる．ページ・テーブルのアドレスが30,000であることから，これにページ番号を加算して該当ページのページ枠アドレス 184,000 を取り出す．このアドレスに仮想アドレスの変位を加算すると実アドレスが算出できるのである．以上のような処理を，ハードウェア機構，つまり動的アドレス変換機構を通じて，命令を実行するつど行なうのである．

このようなセグメンテーション・ページング方式を採用すると，さきのページング方式と異なり，ページ・テーブルのスペースが節約でき，仮想記憶装置間で同一プログラム・コードの共用が容易になる．

たとえば，図 9-26において，仮想記憶装置内のセグメント 3 には，何も入っていない．この場合，セグメント・テーブルの記入項目は必要であるが，その中の標識でページ・テーブルが不在であることを表示することによって，その

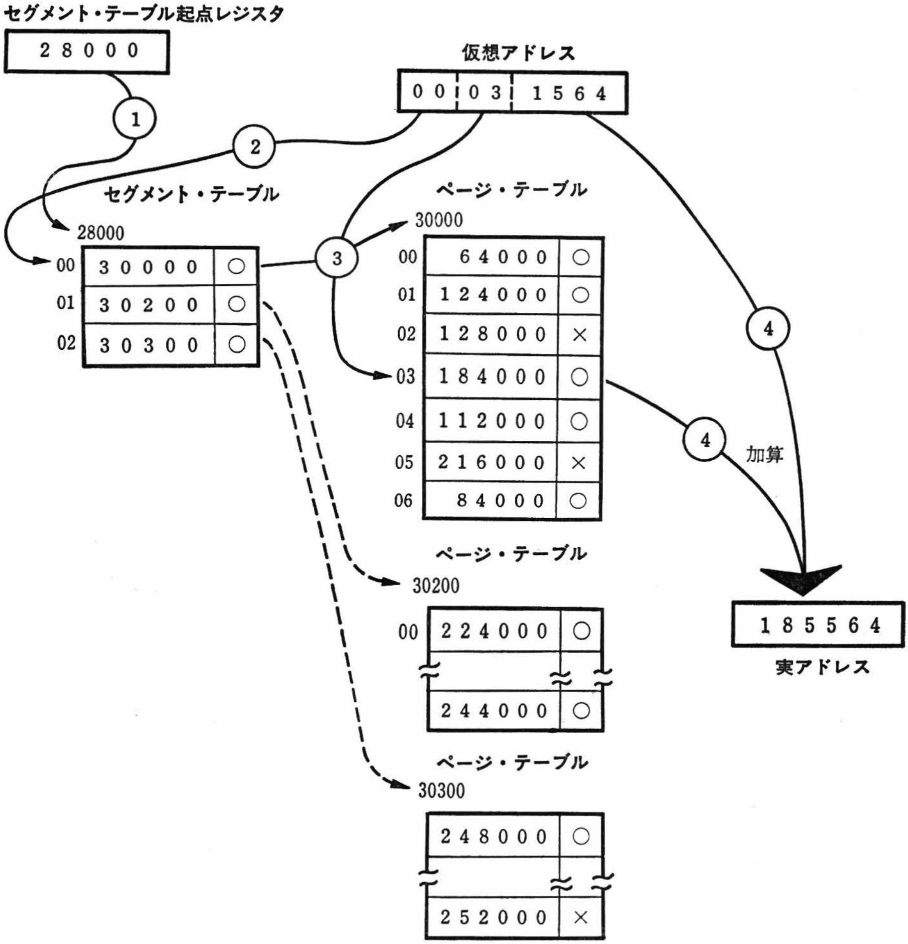


図 9-28　動的アドレス変換の例

ページ・テーブルを作る必要はなくなる．こうすれば，セグメント・テーブルの記入項目の部分は仕方がないとしても，さきのページング方式のように空白のページ・テーブルの記入項目を作らなくてすむ．さらに，セグメント・テーブルの中に該当ページ・テーブルの長さを示す情報を入れるような欄を設けておけば，ページ・テーブルの大きさを実情に合わせて調整することもできる．たとえば，各セグメントが32ページを含む大きさで構成されていたとしても，

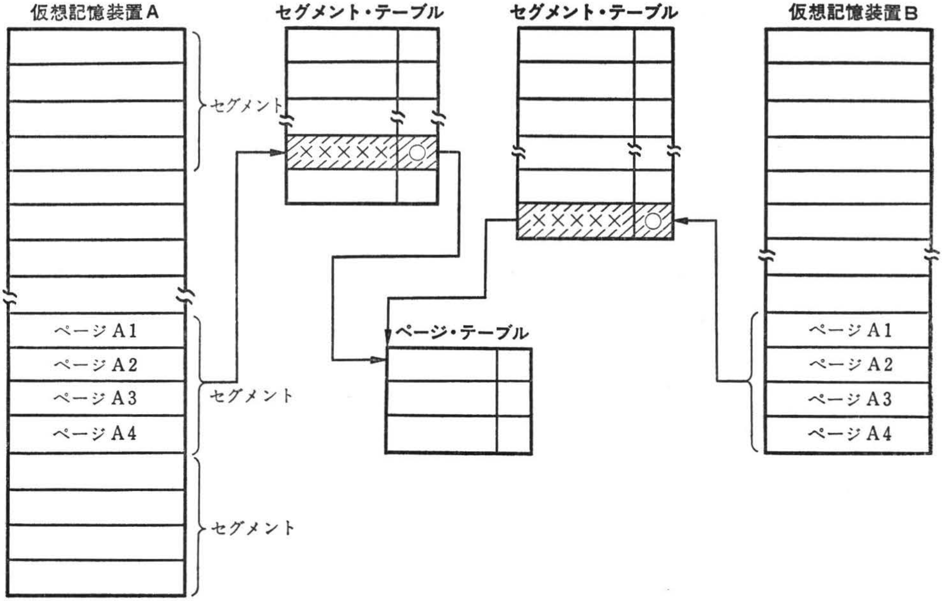


図 9-29　同一プログラム・コードを共用する場合

特定セグメントは12ページしか使われていないとすると，そのページ・テーブルは12個の記入項目ですますことができる．このようにして，ページ・テーブルに要するスペースが節約できる．

1つのシステムに複数個の仮想記憶装置も設けることを，**多重仮想記憶装置**(multiple virtual storage）というが，セグメンテーション・ページング方式だと別の仮想記憶装置との間でプログラム・モジュールを共用することが非常に容易になる．この様子を示したのが，**図 9-29** である．ここに示したように，別々の仮想記憶装置で同じプログラム・コードを共用するときには，それぞれのセグメント・テーブルで同じページ・テーブルを指すことによって，このことが容易に実現できる．このようなコントロールのしやすさは，少ない CPU オーバレッドでページ・コントロールが行なえるという結果を生むことになるのである．

セグメンテーションとページングを用いた動的再配置の効果は，すべてコン

ピュータ・システムのハードウェアとソフトウェアによって得られるのであり，プログラマやオペレータは何らかかかわりを持たなくてすむ．監視プログラムが，アドレス空間をセグメントやページに編成し直し，それにふさわしい仮想アドレスを作り出し，プログラムをローディングするさいにセグメント・テーブルとページ・テーブルを作りあげる．一方，動的アドレス変換機構が，プログラムの実行中に自動的にアドレス変換を行ってくれる．なお，プログラムまたはシステムのアドレス空間を管理するのにはセグメンテーションが，実記憶装置の効率的管理にはページングが適している．このことは，次節以降の説明で除々に明らかになってゆくだろう．

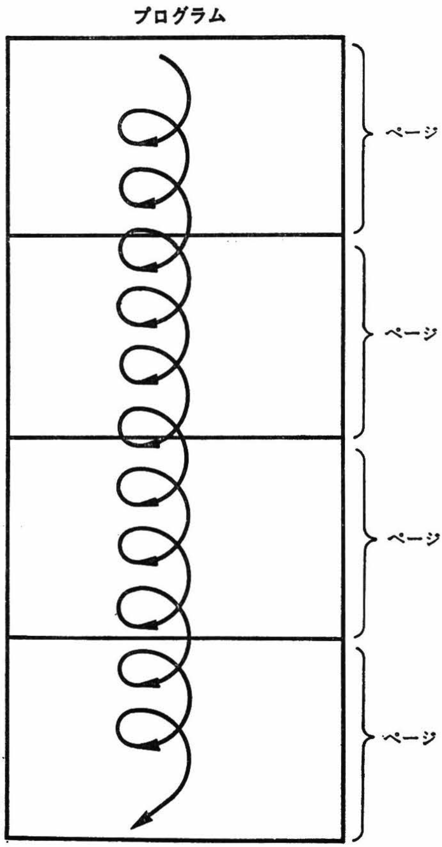


図 9-30　プログラム参照の局所性

2）連想レジスタによる高速アドレス変換

これまで説明してきたようなセグメンテーション・ページング方式で，命令の実行のつど動的アドレス変換を行なうとなると，プログラムの実行時間が長くなるのではないかという問題が生じる．アドレスの参照のたびに仮想アドレスから実アドレスに変換を行なうのであるから，いくらハードウェア機構で行なうといっても大変なオーバーヘッドであることは疑いない．ところが，一般にプログラムの参照の仕方には**局所性**（locality）という性質がある．すなわち**図 9-30**で示しているようにプログラムの命令実行の順序は，比較的近接した部分をループしながら先へ先へと進んで行くのが普通である．しかも，最近の**構造的プログラミング**（structured programming）**技法**などでは，極力GOTO命令などを使わない方向にあるので，この傾向はますます強まっている．ともあれ，この局所性という性質を利用すると，**連想レジスタ**と呼ぶ特別なハードウェア機構を設けることによって，CPU のオーバヘッドを大幅に軽減することが期待できる．

連想レジスタは，セグメント・テーブルとページ・テーブルを使った比較的遅い変換を補うための機構であり，実記憶装置のテーブルを参照するよりははるかに速い特殊目的用の高速レジスタから構成されている．レジスタは8個の場合もあるし，16個あるいは128個の場合もある．**図 9-31** では8個の連想レジスタを例にとりあげている．図 9-31 の右側に破線で示しているのが，これまで説明してきたテーブルを用いた動的アドレス変換の経路である．一方，左側の実線で示したのが，連想レジスタによる変換の様子である．

連想レジスタには，実行中のプログラムが最後に用いた8種類のページのセグメント番号とページ番号が入っている．プログラムで仮想アドレスの参照が起きると，仮想アドレスのセグメント番号とページ番号の部分が連想レジスタのセグメント番号，ページ番号と比較されるのである．8個のレジスタの中に仮想アドレスの番号と一致するものがあれば，連想レジスタのページ枠のアドレスに仮想アドレスの変位部分が加算されて，実アドレスが得られる．同時にレジスタの参照ビットが1になる．これら8つのレジスタの内容と，仮想アド

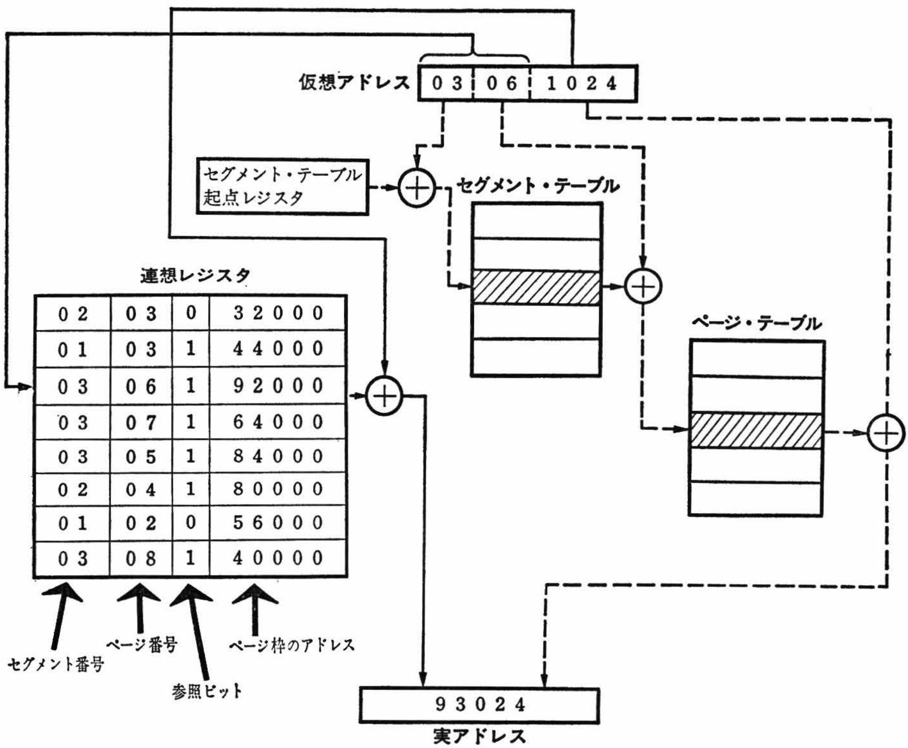


図 9-31　連想レジスタによる動的アドレス変換

レスとの比較は，逐次に行なわれるのではなく全部同時に並行して行なわれる．それだけ短時間で変換できるのである．ともあれ，連想レジスタはきわめて高速であるため，アドレス変換に要する時間はほとんど無視できるほど瞬間的なものである．

仮想アドレスのセグメント番号とページ番号が連想レジスタのいずれにも一致しない場合には，図 9-31の右側に示しているセグメント・テーブルとページ・テーブルを使って変換が行なわれる．実際には，テーブルによる変換は，連想レジスタと仮想アドレスの比較が行なわれるのと同時にスタートする．連想レジスタの内容と仮想アドレスの内容が一致した場合には，テーブルによる変換は直ちに中止されるが，両者が一致しないときはテーブルによる変換が完全

に終了するまで行なわれる．その結果得られたページ枠アドレス，およびセグメント番号とページ番号が連想レジスタのどれかに入れられる．この場合どのレジスタに入れるかは，参照ビットによって決められる．図9-31の連想レジスタの場合には，最初のレジスタと終りから2番目のレジスタの参照ビットが0であるので，このいずれかが使われる．なお，新しい内容が連想レジスタに入る場合にも参照ビットは自動的に1となる．

連想レジスタのすべての参照ビットが1になると，参照ビットのいずれもが0に戻される．その結果，再び使用されたページに対応するレジスタの参照ビットは再び1にセットされる仕組みになっている．このようにして，最も時間的に新しく使用されたページを絶えず連想レジスタに保持するやり方を，**LRU法**（least recently used method）と呼ぶ．連想レジスタにおけるLRU法は，完全にハードウェア機構によって実行される．

以上みてきたように，連想レジスタには実行中のプログラムで最も時間的に新しく用いられた8ページ分のアドレス情報が収容されている．ページの大きさを4Kと仮定すると，最後に参照された32Kバイト分のアドレス変換情報が貯えられていることになる．しかも，プログラムの参照の仕方には，前述したような局所性という性質があるので，ほとんどのアドレス変換は，連想レジスタで行なわれることになる．連想レジスタでの変換は実記憶装置を全くアクセスすることなく，極めて高速で達成されることから，セグメント・テーブルとページ・テーブルを使う変換がひき起すおそれのある効率低下は，完全に回避できるのである．

5. 実記憶装置と外部ページ記憶装置の管理の仕方

前項までは，命令の実行時にアドレスを動的に変換する仕組みについて検討してきた．動的アドレス変換が実現できることによって，仮想記憶装置と実記憶装置の結びつけが達成できる．つまり，仮想記憶装置上にローディングされたプログラムが実記憶装置で実行できるようになる．プログラムは仮想記憶装置上の連続したアドレス空間にローディングされ，仮想アドレスが割り当てられる．しかし，仮想記憶装置は概念上の記憶装置であって，実際に存在するも

のではない．したがって，仮想記憶装置上でプログラムが実行できるわけはない．プログラムは，実記憶装置上のページ枠にページ単位で分散してページインされる．しかも，仮想アドレスをもった状態でページ枠に記憶される．そこで命令が実行されるさいには，この仮想アドレスを実アドレスに変換する必要が生じる．このアドレス変換を受け持つのが**動的アドレス変換機構**（dynamic address translation feature）と**連想レジスタ**（associative register）とである．

セグメンテーション・ページング方式の動的アドレス変換を行なうと，記憶域の断片化は極力排除でき，実記憶装置の有効利用が実現できる．しかも，使用頻度の高いプログラム部分は常時実記憶装置に存在し，逆に使用頻度が低いプログラム部分は，外部ページ記憶装置に記憶され，必要のつどページインされる．これによって，使用頻度に応じた実記憶装置の効率的利用も実現できる．これに関する詳細は後述する．動的アドレス変換がおこなわれることは，単に記憶装置という資源の有効利用だけに貢献するのではない．人的資源であるプログラマにも貢献する．プログラマは仮想記憶装置で与えられるアドレス空間を念頭にプログラムを設計し，コーディングすればよい．仮想記憶装置は，実記憶装置のアドレス空間よりはるかに大きい．プログラマは実記憶装置のアドレス空間に制約されることなく，のびのびとプログラムが作成できる環境となる．

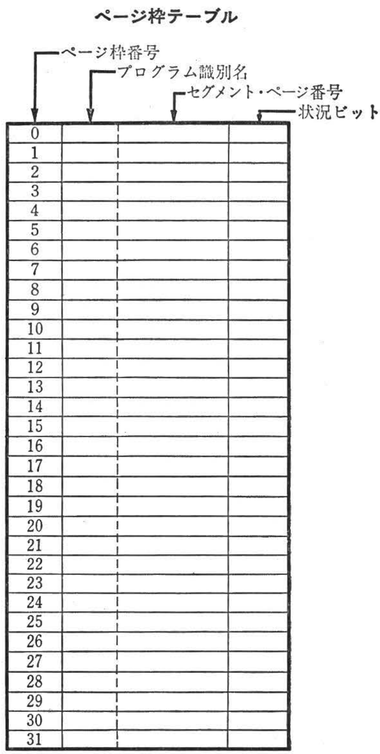
ともあれ，これまでは動的再配置のハードウエアの面ばかりに焦点を合わせて議論してきた．これをどのように利用するかによって，種々のソフトウェアあるいはオペレーティング・システムの作成方法が考えられる．この項では，セグメンテーション・ページング方式のシステムのソフトウエアはどんなことを考慮に入れなければならないかを検討してゆく．

ページングに関連した管理を考える場合，実記憶装置内でのページの割当てと外部記憶装置（外部ページ記憶装置）上のページの割当ての２つについて考えなければならない．前者の管理には，**ページ枠テーブル**（page frame table）を用い，後者には**外部ページ・テーブル**（external page table）を用いる．

1）ページ枠テーブルによる実記憶装置の管理

実記憶装置は，仮想記憶装置上のアドレス空間のページと同じ大きさをもつ**ページ枠**（page frame）に分割されている．ページ枠の大きさは，ページの大きさに合わせて，２Kバイトとか４Kバイト単位に固定されている．**図9-32**の左上に実記憶装置を図示しているが，この例では全体が128Kバイトで，ページ枠の大きさは４Kである．この場合，32個のページ枠が割当て可能である．そこで，オペレーティング・システムでは，これらページ枠をうまく管理して，プログラムにページ単位で実記憶装置を割り振る必要がある．プログラムを構成するそれぞれのページを実記憶装置のページ枠へページインするさい，勝手に任意のページ枠に入れられたのでは困る．別のプログラムやページ





- **ページ枠番号**：ページ枠を識別するための番号．これによって，テーブルをひくことができる．
- **プログラム識別名**：該当ページ枠に入っているページが所属するプログラム名．すなわち，そのページ枠を使用中のプログラムを示す．
- **セグメント・ページ番号**：該当ページ枠に収容されているページのセグメント番号とページ番号を示す．この番号とプログラム識別名とで，ページ枠に入っているページが識別できる．
- **状況ビット**：ページ枠が使用可能かどうかを示す．「0」の場合は，ページ枠が空きの状態であり，「1」の場合は，使用中である．

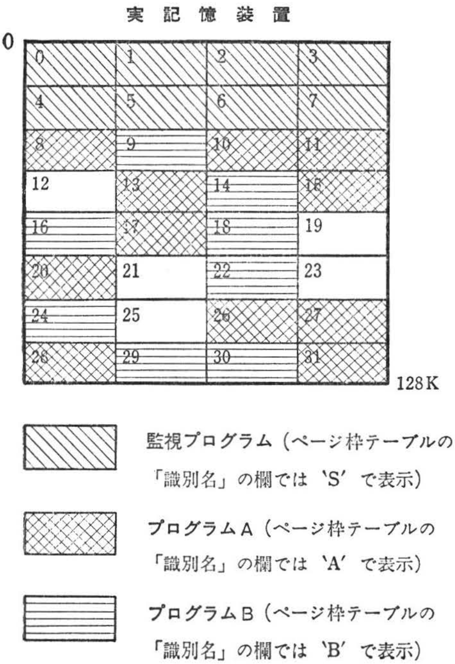
図 9-32　実記憶装置とページ枠テーブル

ですでに使用中のページ枠があるからだ．したがって，各ページ枠が，すでにあるページに割り振られているか，それとも未使用の状態であるかを示す何らかのテーブルがなければならない．このために使用するのが，**ページ枠テーブル**（page frame table）である（**図9-32**参照）．

ページ枠テーブルの大きさは，実記憶装置内のページ枠の数によって決まる．図9-32では，ページ枠が32個あるので，ページ枠テーブルには32個の記入項目が必要である．図で示しているように，ページ枠テーブルの記入項目は，4つの項目，つまりページ枠番号，プログラム識別名，セグメント・ページ番号，状況ビットから構成される．**ページ枠番号**は，実記憶装置のページ枠に対応した一連番号である．それぞれのページ枠にどのプログラムのどのページが収容されているかを示すのが，**プログラム識別名**と**セグメント・ページ番号**である．さらに，該当ページ枠が使用可能かどうかを示すのに**状況ビット**を用いる．

さて，状況ビットに注目して話を進めよう．ページ枠の状況は，状況ビットが“1”（使用中）か“0”（あき）かを見れば判定できる．したがって，制御プログラムが，新しいページを実記憶装置に割り振るときは，ページ枠テーブルの状況ビットが“0”のページ枠を捜し出して使わせればよい．セグメンテーション・ページング方式においては，ページ枠テーブルを使って，ユーザ・プログラムに実記憶装置のページ枠を割り当てる．セグメント・テーブルとページ・テーブルとが動的アドレス変換のためにハードウエアによって使われるのに対し，ページ枠テーブルは実記憶装置を有効に管理してゆくためにソフトウエア（制御プログラム）によって使われるのである．

ここで，プログラムがローディングされたさいのページ枠テーブルの状態を具体例にもとづいて解説しておこう．この例が，**図9-33**である．この図のページ枠テーブルでは，簡略化を意図してページ枠テーブルの中のセグメント・ページ番号は省略している．まず，オペレーティング・システムの中核となる監視プログラムが，実記憶装置に記憶された状態から考えよう．監視プログラムの大きさが32Kバイトで，ページ枠0から8までに連続してローディングされていると仮定しているので，ページ枠テーブルのページ番号0から7までの



ページ枠テーブル

0	S		1
1	S		1
2	S		1
3	S		1
4	S		1
5	S		1
6	S		1
7	S		1
8	A		1
9	B		1
10	A		1
11	A		1
12			0
13	A		1
14	B		1
15	A		1
16	B		1
17	A		1
18	B		1
19			0
20	A		1
21			0
22	B		1
23			0
24	B		1
25			0
26	A		1
27	A		1
28	A		1
29	B		1
30	B		1
31	A		1

図 9-33 プログラムがロードされたときのページ枠テーブルの状態

記入項目の状況ビットはすべて“1”（使用中）となる．

つぎに，ユーザ・プログラムAとBとが記憶された状態を追加してみよう．プログラムAは11個のページ枠，プログラムBは8個のページ枠を使っている．したがって，それぞれのページがロードされているページ枠を忠実に反映したページ枠記入項目が作りあげられる．各々のプログラムを構成しているページは，ページ・テーブルの中でマップされているので，実記憶装置内では連続したページ枠に収容される必要は何らない．ともあれ，ページ枠テーブルをみれば，空きのページ枠がどこに存在し，使用中のページ枠がどのように割当てられているかということがすぐにわかるのである．

なお，プログラムAの実行が終ると，プログラムAのページで使用中であったページ枠テーブルの状況ビットはすべて“0”に戻される．プログラムBが

終った場合も同様のことがおきる．別のプログラムが実記憶装置にローディングされる場合とかページインされる場合，ページ枠テーブルの状況ビットを使ってページ枠を割当てる．このようなページ枠テーブルの概念を導入して，実記憶装置を管理してゆけばよいことになる．

2）外部ページ・テーブルによる外部ページ記憶装置の管理

セグメンテーション・ページング方式による仮想記憶システムに関する説明をおこなったさい，実行中のプログラムを構成するすべてのページは，実記憶装置ないしは外部ページ記憶装置のいずれかに記憶されなければならないことを指摘した．

外部ページ記憶装置は，**ページング・データ・セット**（paging data set）として構成され，実記憶装置上に記憶できないページのコピーが貯えられる．ページング・データ・セットのレコード域は，**スロット**（slot）と呼ぶ．したがって，スロットの大きさは，ページの大きさに等しい．外部ページ記憶装置には，実記憶装置上にない仮想記憶装置のページが存在するが，仮想記憶装置と外部ページ記憶装置との間でのマッピングの方法にはいくつかの種類が考えられる．その1つは，双方の記憶装置を1対1の対応でマッピングする方法である．これは，もっとも単純な方法であるが，仮想記憶装置の数が多くなると膨大なスペースを必要とすることや，磁気ディスクや磁気ドラムのメカニズムを考慮に入れると，動的に書き込み場所がきめられないために書き込むさいに必ず1回転の待ちが生じてしまうといった限界がある．動的に書き込む場所，つまりスロットを決定できるようにすると，仮想記憶装置と外部ページ記憶装置とは1対1の対応ではなくなるが，マッピングは特別なテーブルを設けて保守すればよい．この方法を採用すれば，ダイナミックにページアウトすべきスロットを選定できるので，磁気ディスクや磁気ドラムの特性に応じたもっとも効率的なデータ転送のスケジューリングをおこなうことができる．また，効率を高めるためには，ページング・データ・セットには磁気ドラムを適用するとよい．しかし，すべての仮想記憶域を収容するのにはあまりにも割高になるという場合には，ページの参照頻度によって，磁気ドラムから磁気ディスクへ配置

換えするという方法も考えられる．この手法を**マイグレーション**（migration）と呼んでいるが，ページの選択配置が動的におこなうことができれば，マイグレーションも可能である．また，動的なマッピングが実現され，ページアウトすべき場所がダイナミックに決定されることによって，仮想記憶装置上での記憶域の断片化は事実上何らの損害にもならないことになる．

話をもとに戻そう．先述の実記憶装置の管理の仕方を検討したさいには，実記憶装置にうまく収容できるプログラムについてだけしか考えなかった．これは，仮想記憶システムの概念からすると現実的ではない．ページの何割かは，実記憶装置のページ枠に収容しきれないことを前提にすべきである．これを，出発点とすべきである．このようなシステムでは，マッピングに関する今迄の概念を前述したように拡大しなければならない．このことは，ページ・テーブルに新たな追加項目を設ける必要があるし，新しいタイプのテーブル，つまり**外部ページ・テーブル**（external page table）の概念を必要とする．

まずは，ページ・テーブルへの新たな追加項目について考えよう．もっとも，この項目は，すでに「動的アドレス変換の方法」の個所で標識項目として若干の説明を加えているので全く新しい項目とはいえない．ともあれ，**図 9-34** を参照していただきたい．

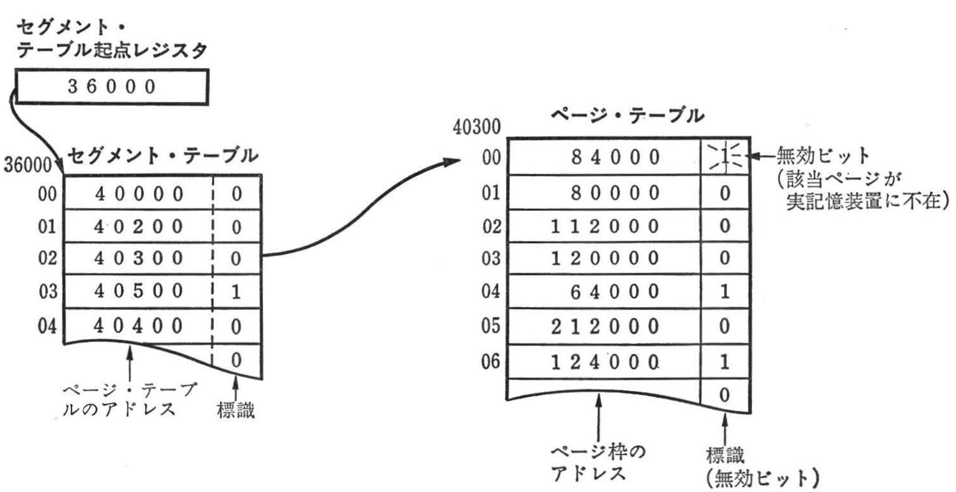


図 9-34 ページ・テーブルの標識（無効ビット）

ページ・テーブルの標識項目に注目して欲しい．このビットは一般に**無効ビット**（invalid bit）と呼ばれている．このビットが“1”であると，該当ページが「無効である」ことを意味する．つまり，そのページは実記憶装置内のページ枠には存在しない．逆に，無効ビットが“0”だと該当ページはページ枠に存在することになる．ともあれ，該当ページがページ枠に存在しないことを，**ページ不在**（page fault）という．ページ不在であることは，不在となったページが，外部ページ記憶装置にページアウトされていることを暗示している．ここで問題となるのが，該当ページがどのスロットに貯えられているかを判定するための手だてである．このために用いられるのが，外部ページ・テーブルである．

外部ページ・テーブルは，**図9-35**で示しているようにそれぞれのページがどのスロットに貯えられているかの関係づけをおこなう．したがって，この外部ページ・テーブルは，ページ・テーブルごとに対応して用意される．外部ページ・テーブルは，外部ページ記憶装置に記憶されているすべてのページに関するスロット場所をマッピングしている．したがって，ページ不在が発見されたさい，制御プログラムは外部ページ・テーブルを用いて，つぎの手順で記憶装置の管理をおこなうことになる．

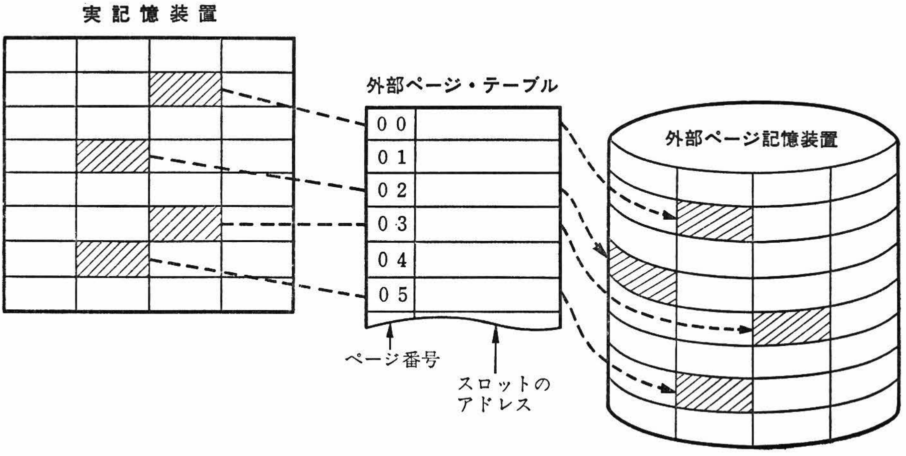


図 9-35　外部ページ・テーブル

① システムは，外部ページ・テーブルの対応記入項目を調べ，外部ページ・テーブルのスロットのアドレスを用いて，外部ページ記憶装置上の該当ページを捜す．

② そのページを収容するための，実記憶装置上の空きのページ枠を選定する．

③ 外部ページ記憶装置上のスロットにある該当ページを，実記憶装置上のページ枠にページインする．

④ ページ・テーブル上の該当記入項目のページ枠アドレスと標識とを更新し，そのページが実記憶装置に存在することを示す．

⑤ 仮想アドレスを実アドレスに変換し，プログラムの実行を再開する．

これまで，セグメンテーション・ページング方式を採用した仮想記憶システムにおいて必要となる各種テーブルについて検討してきたが，この外部ページ・テーブルで必要となるすべてのテーブルが揃ったことになる．ここで，これまで説明してきた各種テーブルの相互関係を把握し，システム全体をまとめてみることにする．これを示したのが**図 9-36** である．これによって，仮想記憶システムを作動させるにあたって必要となる種々のテーブルがどのように使用されるかが全体として把握できよう．

図 9-36 を追跡しておこう．プログラムの中のある命令を実行しようとしたさい，右上に示した仮想アドレスを参照したとしよう．動的アドレス変換機構は，この仮想アドレスのセグメント番号とセグメント・テーブル起点レジスタにある起点アドレスをもとに，セグメント・テーブル上の該当記入項目をつきとめる．さらに，記入項目内のページ・テーブルのアドレスと仮想アドレスのページ番号をもとに，ページ・テーブル上の該当記入項目を決定する．該当記入項目の無効ビットが“1”なので，要求するページが実記憶装置には存在しないことを知る．ここで，ページ不在の割込みが生じそのあとの処理はソフトウエアに移る．ページ監視プログラムは，外部ページ・テーブルを参照し，要求したページが外部ページ記憶装置上のどのスロット場所にあるかをつきとめる．一方，ページ枠テーブルを参照して，あいているページ枠を捜し出し，そ

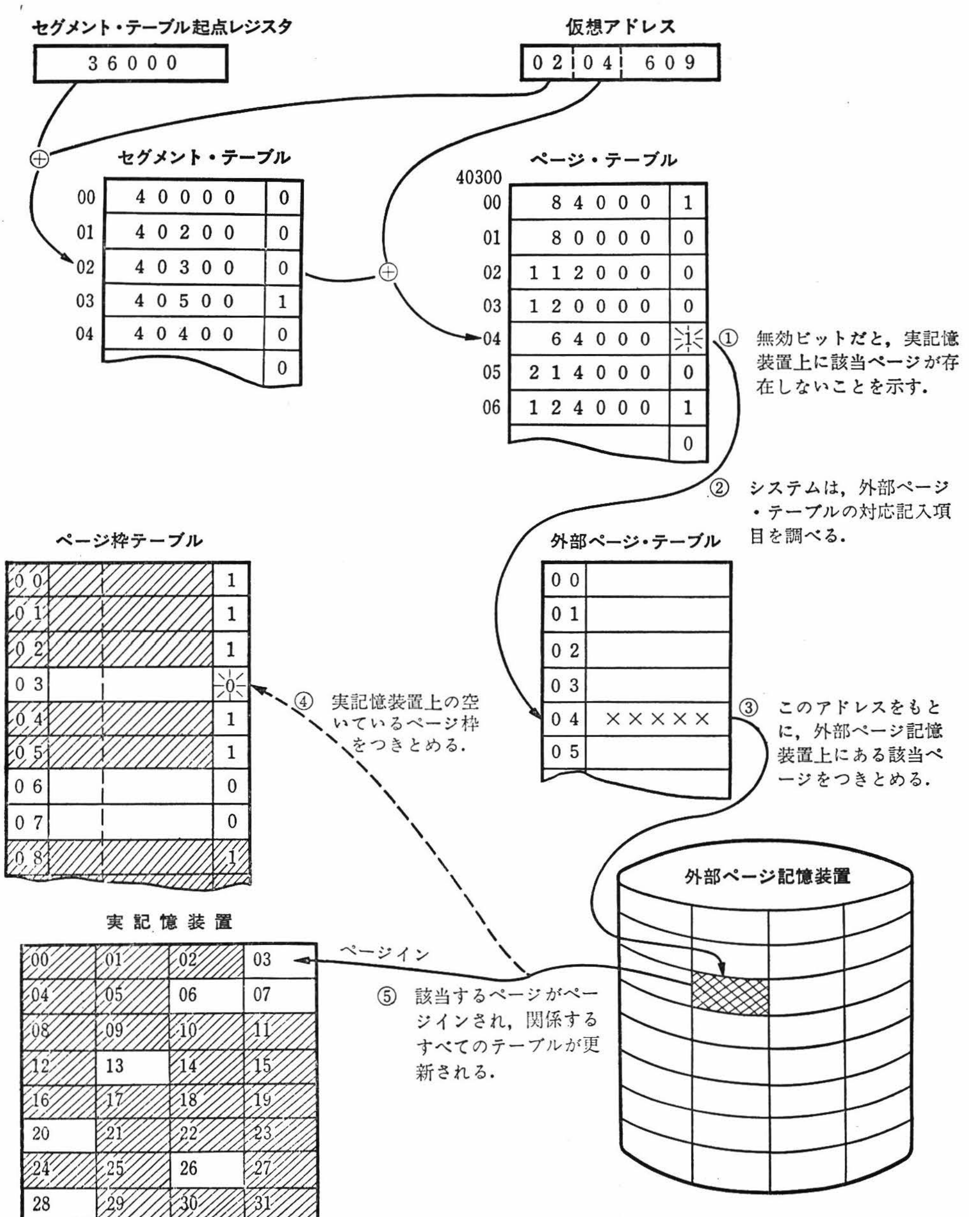


図 9-36　ページ不在の処理

のページ枠へスロットにあるページを移す．これが**ページイン**である．これで要求したページが使用可能となる．ところで，監視プログラムの仕事はこれで終るわけにはいかない．ページ・テーブルの中の該当記入項目をたったいまページインした状態に更新する必要がある．つまり，該当記入項目のページ枠アドレスと無効ビットを更新する．また，ページ枠テーブルの該当記入項目も更新する．そのあとで，仮想アドレスの変換を動的アドレス変換機構にゆだねるのである．

6. 要求時ページングとプログラムの実行

1) 要求時ページングとページ置換

要求時ページング（demand paging）とは，該当ページが必要になった時点にページインするやり方である．前述したように，アドレス変換の必要性が生じたとき，セグメント・テーブルを経由してページ・テーブルを見に行き，ページ・テーブルの記入項目がそのページが実記憶装置上にないことを示しているとき，ページ不在の割込みを起こして外部ページ記憶装置から要求するページを持ってくるやり方である．たいていの仮想記憶システムにおいては，大体この考え方を採用している．しかし，プログラム中のページの要，不要については，論理の流れによってある程度予測がつけられる．したがって，プログラマ自身，あるいはコンパイラが目的モジュールを作るさいに，それらの事前情報をシステムに知らせるような命令を加えてやるということも考えられる．こうすることによって，ページングの不要な回数を減らすことが期待できる．この種のやり方を，**プリ・ページング**（pre-paging）と呼んでいる．しかし，プログラマやコンパイラがそのための命令を加える必要があることのため，実際にこの手法を採用しているシステムはきわめて少ないようだ．したがって，本書では，要求時ページングを前提にこれ以降の説明を続けてゆく．

ページ不在の処理は，前項でふれたようにいくつかの段階を経る必要がある．しかし，利用可能なページ枠が存在するかぎり，その処理は比較的簡単である．前項での説明は，空きのページ枠があることを条件としていた．しか

し，仮想記憶システムでは，仮想記憶装置の方が主記憶装置よりもはるかに大きいのであるから，最終的には利用可能なページ枠はすべて使い果されてしまう．実記憶装置のページ枠が種々のプログラムのページで一杯になっていて，新しいページがページインできないというとき，何らかの法則に従って，いま入っているページのどれかを外部ページ記憶装置に追い出して，新参のページにページ枠をあけ渡してやらなければならない．この操作のことを，**ページ置換**（page replacement）**操作**という．ページ置換は，つぎの2つの理由でシステム・パフォーマンスに大きく影響するので，非常に重大な関心事となる：

① 頻繁に参照されるページを置換すると，すぐにそのページを実記憶装置内のページ枠に戻す必要が生じてしまう．このことは，不必要なシステム・オーバヘッドをもたらすことになるので，極力回避する必要がある．

② ページインされた以降に，そのページの内容がなんらかの変更をうけた場合には，まずページアウトをおこなう必要がある．つまり，何らの変更も加えられていない場合だと，外部ページ記憶装置にあるコピーを必要となった時点にそのままページインしても支障はないが，内容変更をうけたページの古いコピーはすでに使い物にならない．したがって，変更されたページを外部ページ記憶装置にとっておかなければならない．

これら2つのことを念頭にページ置換の方法を検討してみよう．ページ置換の方法にはいくつかが考えられる．まず，**FIFO 置換法**がある．一般に**先入れ-先出し法**（First-In First-Out method；略して **FIFO 法**）と呼ばれる方法である．実記憶装置に最初に入ったページを最初に追い出す方法である．この方式は，実記憶装置の中にもっとも長い間貯えられているページから置換しようという考えである．この方法だと，①で指摘された問題は解決しそうにもない．それに，②の問題に対処するには，ページ内容の変更の有無によって置換の優先順位を検討してゆくことが要求される．

理想的には，将来のページ参照を予知し，頻繁に参照されると予想されるページの置換を極力回避し，逆に二度と参照されないとか，当分参照されること

がないと予想されるページから置換すればよい．しかし，システムに起こる事柄を予知することは困難なので，この理想的な置換法は使用性に乏しいと云わざるを得ない．この理想的な置換法に近いものとして，**LRU** (Least Recently Used) **法**がある．

LRU 法では，長時間参照されたことがないページ，つまり現時点に近い時点でほとんど使用されていないページを最初に置換しようというものである．換言するなら，LRU 法とは，もっとも最近参照されたページ群は実記憶装置のページ枠にとどめておこうとするものである．われわれ人間の世界においても，近い過去に記憶したり理解したことほど，よく参照されるといわれている．たとえば，**図 9-37** で，A時点で記憶したものよりもC時点で記憶したものの方が，近い将来に参照されたり，利用されたりする可能性が高い．このことは，人間の世界にとどまらず，プログラムにおいても十分にあてはまる．特に，**局所性** (locality) という性格をもつプログラムにおいては，人間世界におけるよりもはるかに LRU 法があてはまると想定される．ともあれ，LRU 法では現時点に近い時点で使用されたページ群は，近い将来再び参照を受けるはずだという仮定にもとづいている．これによって，頻繁に参照されるページ群を極力実記憶装置のページ枠にとどめておくようにコントロールできる．

これまでのところでは，参照されたページや変更されたページをシステムがどのようにして識別するかという点について何ら言及していない．そこで，これについて補足しておこう．実は，これらは各ページ枠単位に設けられている記憶保護キーの中の2つの特別なビットによって制御される仕組みになっている．1つは，**参照ビット** (reference bit) と呼ばれ，もう1つは**変更ビット** (change bit) と呼ばれている．参照ビットは，その記憶域が使用されたかど

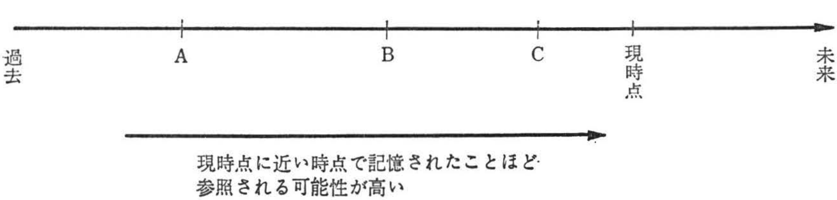


図 9-37 LRU の基本的な考え方

うかを表示する．つまり，該当ページ枠内の内容が参照されたり，ページ枠の中に記憶されたりしたときは，いつでも自動的に“1”ビットに変えられる．一方，変更ビットはその記憶域の内容が変更されたかどうかを示す．つまり，データがページ枠内のどこかに貯えられると，自動的に“1”ビットになる．この変更ビットは，ページアウト操作が必要かどうかを判定するのに使用される．なお，これら参照ビットと変更ビットとは**図 9-38** に示しているように，ページ枠テーブルの新たな項目として追加される．

さて，以上の基本的事項をふまえて，ページ置換を必要とするページ不在の状態にいかに対処してゆくべきか検討してみよう．

ページインのオペレーションは，ページ置換を必要としようがしまいが何ら変りはない．しかし，ページ置換を必要とする場合には，まず置換すべきペー

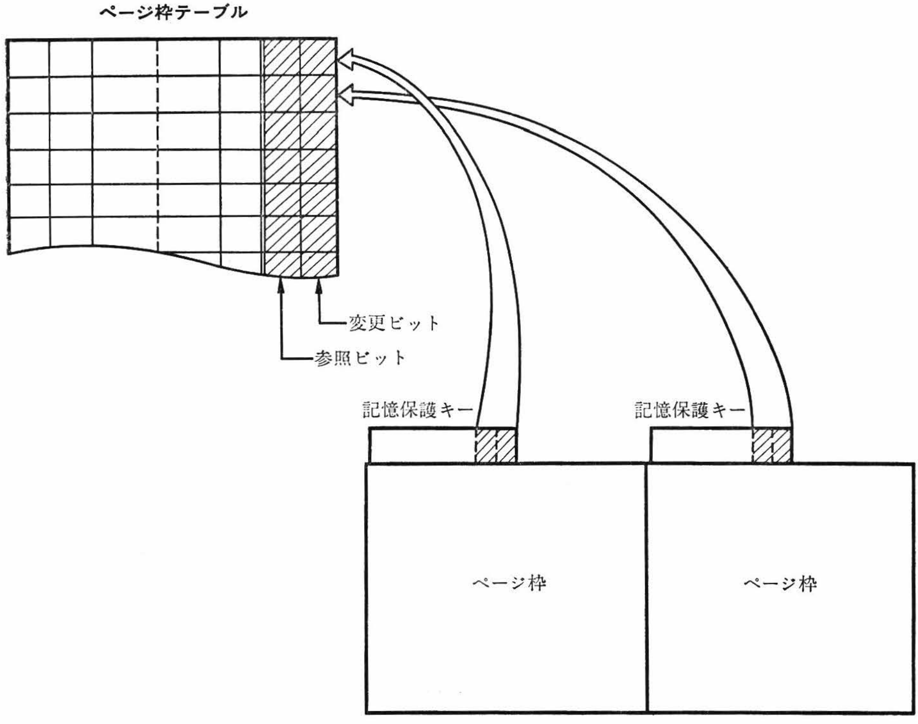


図 9-38　記憶保護キーと参照ビットおよび変更ビットの関係

ジを選択する必要がある．この置換作業は，いうまでもなく監視プログラムで面倒をみる．しかも，多くの場合 LRU 法にもとづいてページ置換を遂行する．そこで，2つの場合をとりあげてLRU 法にもとづいたページ置換のアルゴリズムを考えてみよう：

① 置換の対象となるページが，ページインされて以降なんら変更されていない場合．
② 置換の対象となるページが，ページイン以降すでに変更されている場合

まず，①の場合から検討しよう．この場合には，これまでに説明したページ不在を処理する手順に特に新しいステップを追加する必要はない．主な処理過程は，つぎの通りである．

仮想アドレスの変換中に，ページ不在が発見される．必要とするページが実記憶装置にないため，そのページに対応するページ・テーブルの記入項目の不在ビットがオンになっていることで分る．システムは，ページ不在を処理するために，プログラムの実行を中断しページ不在割込みをひき起こす．そのあとで，まず，最初に実記憶装置のページ枠をみつけ出す必要がある．このために監視プログラムはページ枠テーブルを検査する．もし，利用可能なページ枠があればそれを利用する．ただし，目下の話題は，すべてのページ枠がふさがっている場合である．したがって，図 9-39 で示してるるように状況ビットはすべて“1”の状態を前提にしている．この場合，システムは LRU 法にもとづいて置換すべきページを選択する．①の場合には，ページの内容はページイン以降なんら変更されていない．このことは，ページ枠テーブルの変更ビットが“0”の状態で表示されている．この状態の場合には，外部ページ記憶装置に置換の対象となっているページのコピーが貯えられているので，ページアウトの操作は不要である．置換されるページのページ・テーブルの該当記入項目中の無効ビットをオンにするだけである．このあとで，監視プログラムはページ不在を引き起したページをページインし，該当するテーブルを更新しプログラムの実行を再開するようにすればよい．

一方，②の場合には，置換対象のページの内容がすでに変更されているの

ページ枠番号	プログラム識別名	セグメント・ページ番号	状況ビット	参照ビット	変更ビット
01			1	1	1
02			1	1	0
03			1	1	0
04			1	1	1
05			1	0	1
06			1	1	1
07			1	1	1
08			1	0	0
09			1	1	0

（01）現時点に近い時点で参照されており，かつページインされたのちにページ内容が変更されている．

（03）現時点に近い時点で参照されているが，ページインされたのちにページ内容は変更されていない．

（05）現時点に近い時点ではまだ参照されていないが，ページインされたのちにページ内容が変更されている．

（08）現時点に近い時点ではまだ参照されておらず，またページインされたのちにページ内容の変更もされていない．

変更ビット：0：変更されていない　1：変更されている

参照ビット：0：参照されていない　1：参照されている

図 9-39　ページ枠テーブルの参照ビットと変更ビット

で，ページアウト操作によって，そのページのコピーをあらためて保存する必要がある．こうしないと，置換されたページが再び参照される場合，古いコピーがページインされることになり，プログラムの最新の状態を反映しなくなってしまう．ページアウトの操作は，実記憶装置のページ枠から外部ページ記憶装置のスロットへ，該当ページを転送することである．この場合，選択されるスロットは，そのページの古いコピーをおさめているスロットである必要はない．必要なことは，新しくページを収容したスロットの場所を指示するために，外部ページ・テーブルの該当記入項目を更新することである．このようにして，ページアウトを遂行したあとで，ページ不在を起したページをページインし，関連するテーブルの記入項目を更新してから，プログラムの実行を再開すればよい．

この種の操作を参照ビットが“0”のものから優先しておこなうことによって，LRU 法にもとづいたページ置換が実現できるのである．

以上みてきたような要求時ページングと効率のよいページ置換のアルゴリズムによって，実記憶装置の断片化が防止できるだけでなく，もっともよく参照

されるプログラムのもっともよく利用される部分が実記憶装置にロードされている状態が常に保たれることになる．単にプログラムに対して大きなアドレス空間を与えるだけでなく，このような方法で主記憶装置資源の最大限有効活用をはかろうと意図しているのである．

2） プログラムのローディングと実行

これまで，ページ枠テーブルをもとにした実記憶装置の管理の仕方と外部ページ・テーブルを用いた外部ページ記憶装置の管理の仕方をふまえたうえで，要求時ページングとページ置換の基本的考え方を検討してきた．ここでは，これまで説明してきた仮想記憶システムの基本的な構造をまとめる意味で，プログラムのローディングの方法とその実行にさいしてのページ不在の処理手順について解説する．

ローディングされるプログラムは，外部記憶装置上のプログラム・ライブラリにある．このライブラリ自体は，仮想記憶システムを用いない場合のプログラム・ライブラリと全く同じである．プログラムは仮想記憶装置を対象にローディングされるが，この場合のアドレスは**静的再配置**（static relocation）である．従来は主記憶装置に対して静的再配置を行なっていたのであるが，仮想記憶システムでは，主記憶装置の代りに仮想記憶装置に静的再配置をおこなうのである．基本的には，主記憶装置に静的再配置をおこなうのと変りないが，この再配置は以前のシステムでは実アドレスを使っておこなわれたのに対し，仮想記憶システムでは，仮想アドレスを用いて再配置をする．

プログラムは，ローディングという入出力操作によって，ページ単位に実記憶装置にもってこられる．実記憶装置内に使用可能なページ枠が十分あれば，すべてのページが実記憶装置にとどまる．この場合でも，すべてのアドレスが仮想アドレスに再配置されるのであり，実記憶装置のページが収容された場所の実アドレスとは何ら関係しない．実記憶装置内に十分なページ枠がないと，はみ出たページは外部ページ記憶装置に書き出される．なお，プログラムのローディングのさいに，セグメント・テーブルとページ・テーブルが作りあげられる．以下でプログラム・ローディングの具体的様子をみてゆこう．

まずは，**図 9-40** を参照していただこう．この図で実記憶装置のページ枠に斜線がほどこされている部分は，該当ページ枠がすでに別のプログラムページで使用されていることを表示している．つまり，ページ枠テーブルの対応する記入項目の状況ビットが“1”であることを意味している．ともあれ，図では，目下プログラムＢをローディングしようとしている．プログラム・ローダーはプログラムＢをプログラム・ライブラリから捜し出し，そのローディングを開始する．ローディングは仮想記憶装置を念頭におこなわれる．この例では，仮想記憶装置上の2000Ｋバイト目から始まる**区域**(region)にローディングしようとしている．なお，仮想記憶装置はセグメント単位でプログラムに割り振られる．**図 9-40**では，プログラムＢの最初のページがローディングされようとして

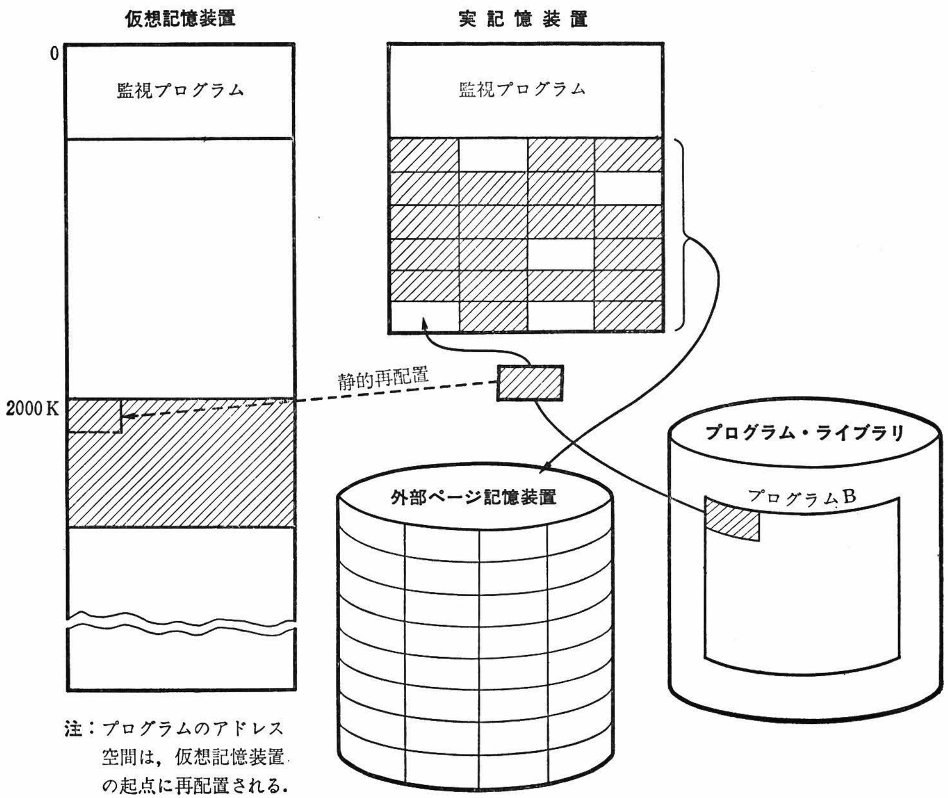


図 9-40　プログラムのローディング　(1)

いる．当然のことながら，2000Kバイトを起点とした静的再配置がおこなわれる．プログラムの中で参照しているすべてのアドレスに対して仮想アドレスを割り振る．そのあと，プログラム・ローダーは，ページ枠テーブルの状況ビットをチェックし，空きのページ枠があるとそこに該当ページを入れる．例では，20番目のページ枠にローディング中のページを記憶しょうとしている．いうまでもなく，このページは2000Kバイト目から仮想記憶装置にローディングしたのであるから，ページ内の参照アドレスはすべて2000Kバイト目を起点とした仮想アドレスに調整されたものである．

プログラムBの最初のページがローディングされると，ひきつづいて2ページ目，3ページ目とローディングされてゆく（図9-41参照）．仮想記憶装置に

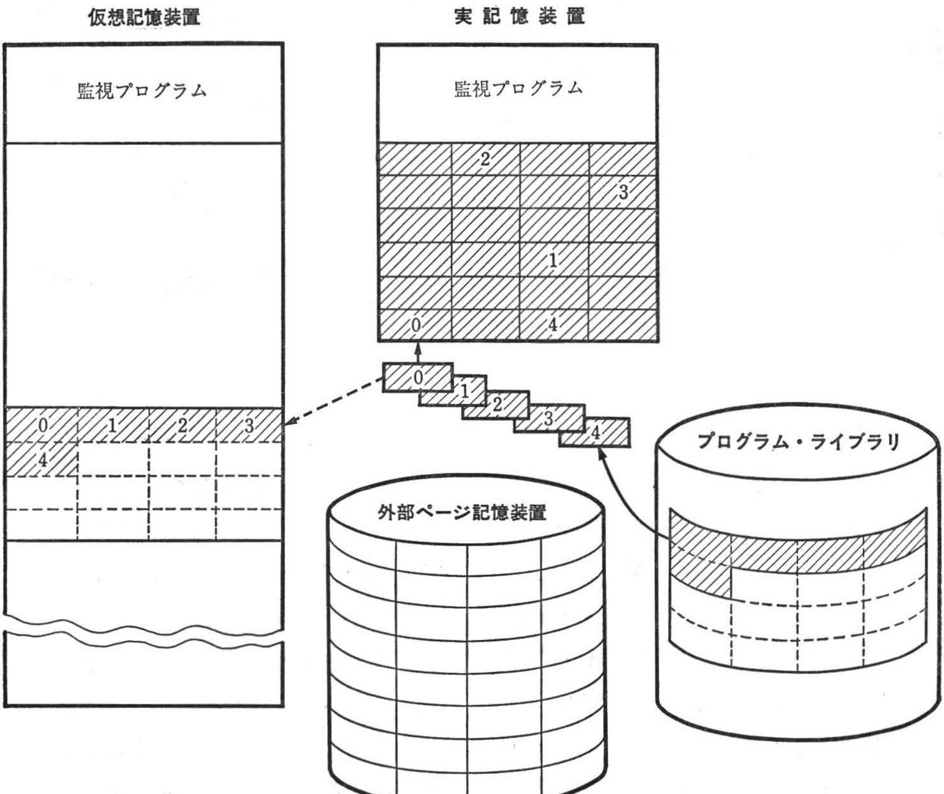


図 9-41　プログラム・ローディング　(2)

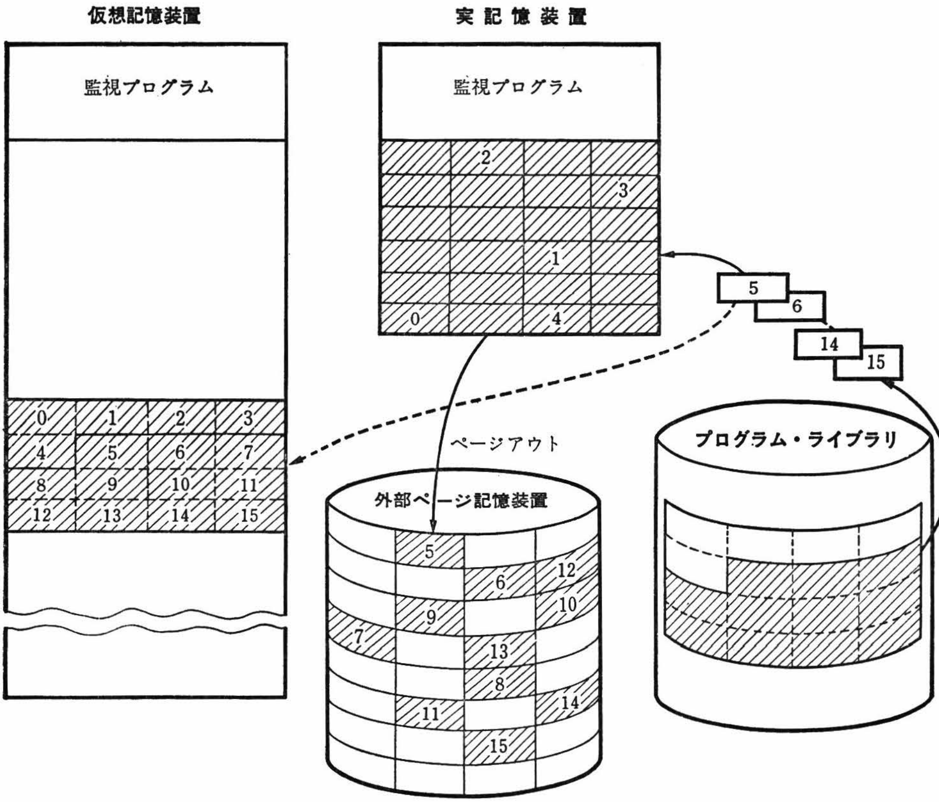


図 9-42　プログラム・ローディング　(3)

静的再配置されると同時に，実際のページは，実記憶装置に空きのページ枠が存在するかぎり，その場所に貯えられてゆく，なお，ページ枠にページが収容されると，それを反映してページ枠テーブルの対応記入項目が更新されてゆく．つまり，プログラム識別名とセグメント・ページ番号が書き変えられ，さらに状況ビットがオンにされる．

図 9-41 では，すべてのページ枠が使用されてしまった状態を示している．したがって，後続するページはページ枠に収容することはできない．**図 9-42** で示しているように，後続するページは外部ページ記憶装置上の空きのスロットにはき出されることになる．この場合にも，仮想記憶装置へのローディングはこれまでと全く同じである．ページ枠テーブルの状況ビットを調べ，すべて

のページ枠が使用中であることが判明すると，外部ページ記憶装置上の空きのスロットに該当ページをコピーする．このページが必要になったさいのページインに備えるためである．いずれにしろ，スロットに書き出しをおこなうと，それを反映して，外部ページ・テーブルの該当記入項目を更新する．つまり，そのページを収容したスロットのアドレスを記入項目の所定欄に入れる．

このようにして，プログラムのローディングを遂行してゆくのである．要は，仮想記憶装置上へのローディングは，従来の主記憶装置へのローディングと本質的に変らない．従来の主記憶装置を仮想記憶装置に置きかえて考えればよい．ただし，仮想記憶装置は実在しないので，若干の工夫が必要となる．プログラムを仮想記憶装置にローディングしたつもりで，アドレス調整がなされる（仮想アドレスにされる）．仮想記憶装置をマッピングするために，セグメント・テーブルとページ・テーブルを作りあげてゆく．また，ローディングされたプログラムを構成するページはどこかに存在しないと困る．そのため，ページ枠に空きがあればそこに収容するし，不幸にして全てのページ枠が使用中であれば外部ページ記憶装置上のスロットに記憶するという手段をとる．それぞれに応じて，ページ枠テーブルないしは外部ページ・テーブルの記入項目を正しく更新するのもプログラム・ローダーの機能である．

ローディングされたプログラムは各命令毎に実行されてゆくが，多重プログラミングのもとでは，いくつかのプログラムが併行して実行される．また，仮想記憶システムでは，実記憶装置の大きさよりも大きなアドレス空間が与えられることから，従来のシステムよりも多くのプログラムを同時に実行することが可能となり，それだけ実記憶装置のページ枠が不足し，ページアウト／ページインの操作が必要となる．したがって，プログラムの実行中に参照しようとしたページがページアウトされてしまっていて，実記憶装置内にないことは十分ありうる．この場合には，**ページ不在**（page fault）の割り込みが生じ，必要なページを外部ページ記憶装置上のスロットからページインした後，プログラムの実行が再開される仕組みになっていた．

まず，ここでページ不在の割込みが生じたさい監視プログラムがおこなう基本的な処理手順をまとめておこう．これを示したのが，**図 9-43** である．

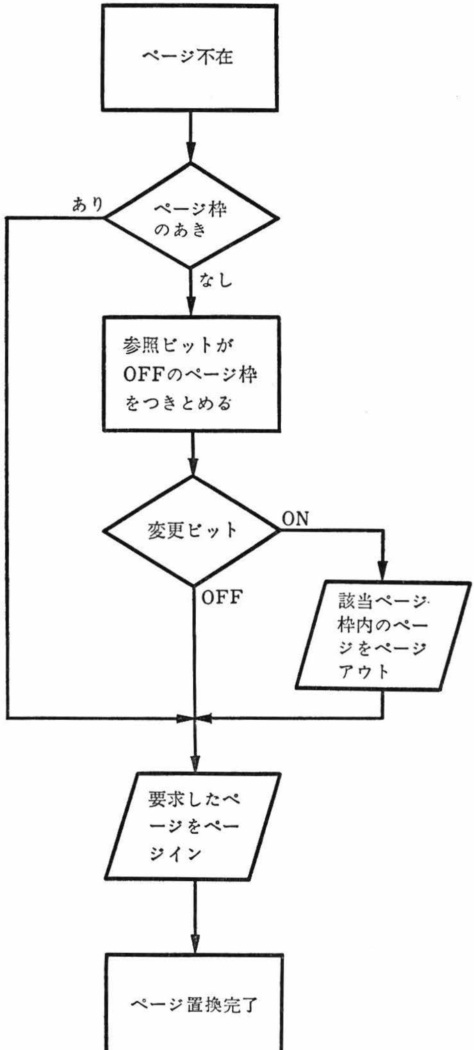


図 9-43　ページ置換の処理の流れ

監視プログラムは，最初にページ枠テーブルの状況ビットを調べる．状況ビットのうちに“0”があれば（つまり，ページ枠にあきがあれば），そのページ枠に該当ページをページインし，ページ・テーブルとページ枠テーブルの該

当記入項目を更新する．ページ枠がすべて使用中であることが判明すると，ページ枠テーブルの中の参照ビットをチェックし，このビットが“0”である記入項目をつきとめる．さらに，その記入項目の変更ビットがオンであるかどうか調べる．変更ビットが“0”であると，そのページ枠に入っているページには何らの変更も加えられていないことになり，外部ページ記憶装置上に入れられているコピーがそのまま使える．したがって，ページアウトの操作は不要である．一方，変更ビットがオンであるとそのページはすでに変更が加えられていることを意味し，外部ページ記憶装置上のコピーはもはや役に立たない．したがって，そのページをあらためてページアウトしたあとで，該当ページ枠に要求ページをページインする．この場合には，ページ・テーブル，ページ枠テーブル，外部ページ・テーブルのいずれも更新される．

基本的な手順は以上の通りであるが，ページインのさい実記憶装置のどのページ枠を選択すべきか，また外部ページ記憶装置のどのスロットにページアウトすべきかの決定は監視プログラムがおこない，その時点において最適な場所を選択する．なお，これらのアルゴリズムの詳細はそれぞれの監視プログラムでかなり異なる．

プログラムの実行中にページ置換を伴わないページ不在が生じた場合の処理の様子をまとめたのが，**図9-44**である．図中の数字記号をもとにポイントをまとめておく：

① 仮想アドレスで参照したページの無効ビットが“1”になっているので，アドレスの変換中にページ不在の割込みが生じ，ページ監視プログラムに制御権が移る．

② 監視プログラムは，外部ページ・テーブルの該当記入項目を参照して，そのページが記憶されているスロットの場所を捜し出す．

③ ページ枠テーブルの各記入項目を調べる．すべての状況ビットが“1”であるから，さらに参照ビットをチェックする．ページ枠2の参照ビットが“0”であるので，このページ枠を選択したとする．その変更ビットが“0”であることから，このページがすでにスロットに記憶されている内

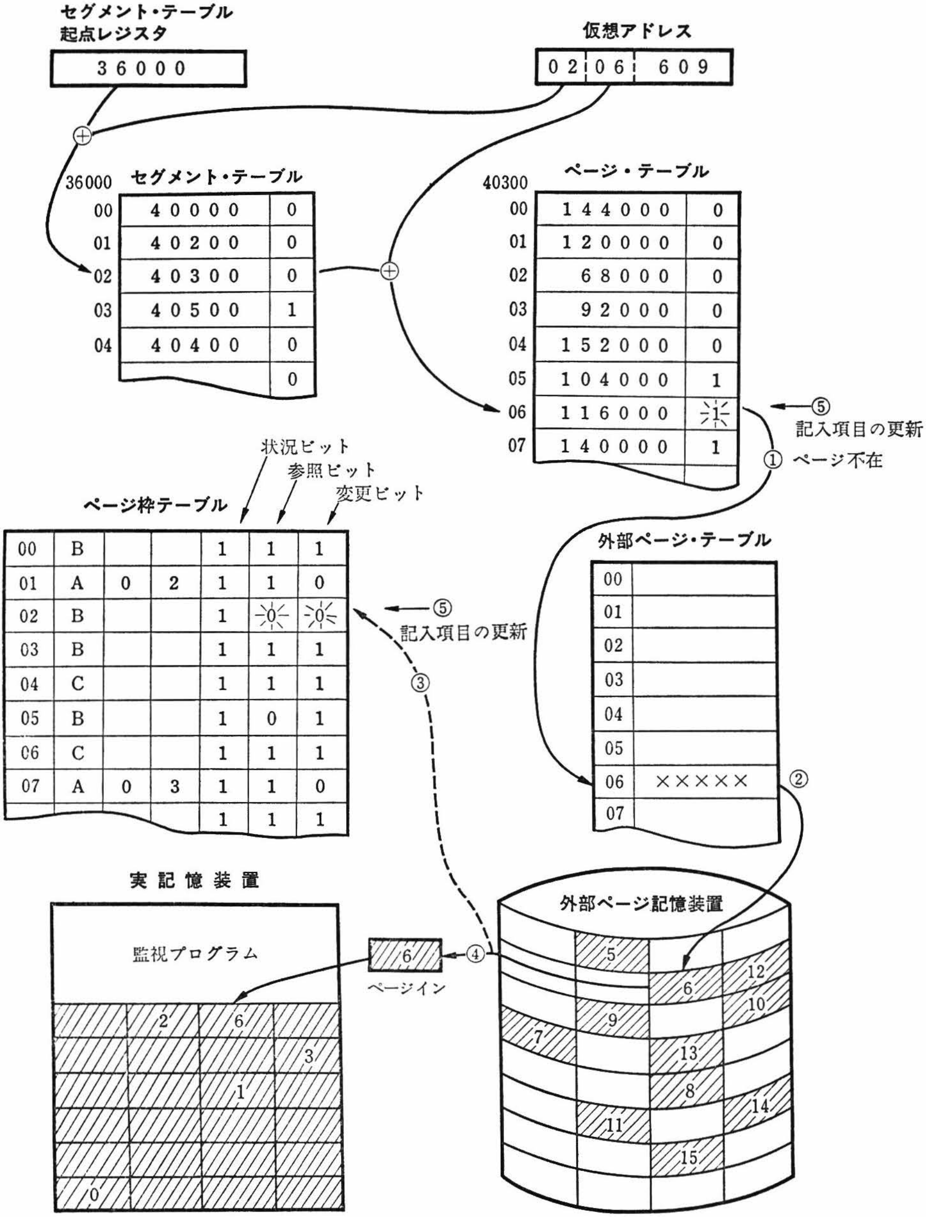


図 9-44　ページ不在の場合のページイン

容と同じであることが判明する．したがって，ページアウトは不要である．

④ 要求ページを，スロットから該当ページ枠にページインする．

⑤ 要求ページをページ枠2にページインした結果を反映して，ページ・テーブルとページ枠テーブルの対応する記入項目を更新する．

以上の処理をおこなったあと，アドレス変換を再開してプログラムの実行が進んでゆく．

一方，プログラムの実行中にページ置換を伴なったページ不在が生じた場合の処理手順を具体化したのが**図9–45**である：

① アドレス変換中にページ不在の割込みが生じ，ページ監視プログラムに制御権が渡る．

② 外部ページ・テーブルの該当記入項目を参照して，要求ページが収容されているスロットのアドレスを得る．

③ ページ枠テーブルの状況ビットを調べる．すべて“1”であることから，参照ビットをチェックし，置換するページ枠5を選ぶ．この記入項目の変更ビットは“1”なので，このページはページイン以降にその内容が変更されていることを意味する．

④ ページ枠5にあるページをページアウトし，それに対応する外部ページ・テーブルの記入項目を更新する．

⑤ 要求ページをスロットからページインする．

⑥ 要求ページをページ枠5にページインした結果を反映して，ページ・テーブルとページ枠テーブルの対応記入項目を更新する．

このあとで，アドレス変換を再開し処理を続行する．なお，すでにプログラム・ローディングの個所で説明したように，すべてのページが最初から外部ページ記憶装置にコピーされているとは限らない．ページ枠に空きがあると，そこにロードされるだけで，スロットには移されていない．この場合，ページ枠の内容がロードされたページで置きかえられることによって，変更ビットが

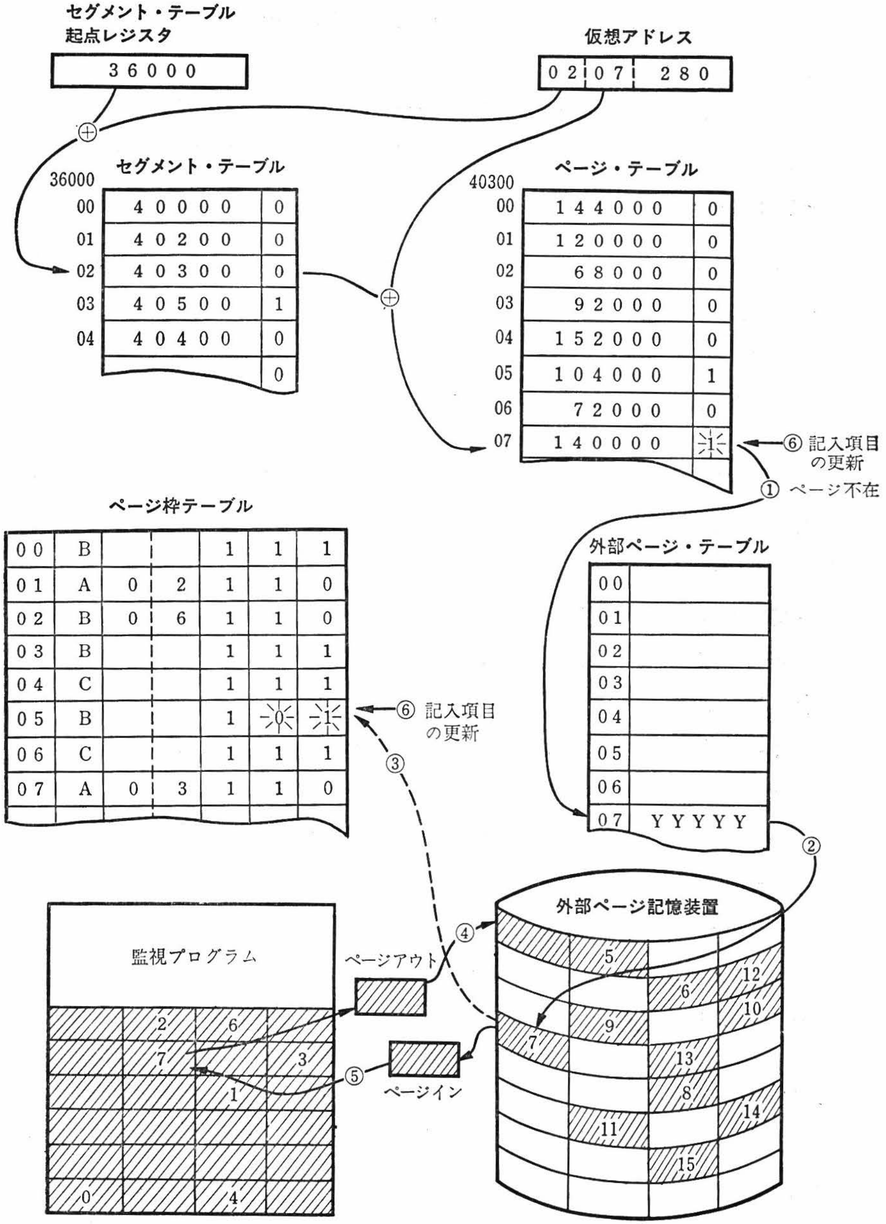


図 9-45　ページ不在の場合のページアウトとページイン

“1”になる．そのため，プログラムの実行が進んでそのページ枠が置換される必要が生じたとき，はじめてページアウトされ，スロットにコピーが記憶されるのである．これ以降は，必要に応じてページイン，ページアウトをおこないながら，変更ビットが設定されるという仕組みになっている．

これまでのような過程を経てプログラムの実行が終了すると，そのプログラムに割当てられた仮想記憶域は解放される．同時に，プログラムの終了時点で割当てられていた実記憶装置のページ枠および外部ページ記憶装置のスロットも解放される．

7. ページの固定化

これまでは，すべてのページがページアウト／ページインの対象になることを前提にして説明してきた．しかし，特別なプログラムだとか，特別なプログラム部分が絶えずページアウトないしはページインをおこなうと不都合が生じてしまう．そこで，ある種のページを長時間ないしは一定の時間ページングの対象にしないで，実記憶装置にとどめておく必要がある．このことを一般に**ページの固定化**（page fix）と呼んでいる．

要求時ページング・システムでページを固定化する理由として，つぎの3つが考えられる：

① システム全体が絶えず必要とする機能は，実記憶装置内に固定化しておく必要がある．たとえば，監視プログラムの中の中核プログラム部分がページアウトされたとしたら，システムのすべてが待ち状態になってしまうであろう．**ページ監視プログラム**（paging supervisor）が，ページング機能の面倒をみているわけだが，もしこのプログラムが実記憶装置に固定されていないとしたら，結局他のプログラムのページングができなくなってしまうわけである．それに，きわめて使用頻度の高い監視プログラム部分がページングされたのでは，それだけシステムのオーバヘッドが高まってしまう．これも回避した方がよい．

② あるオペレーションが持続している間に，そのオペレーションで使用し

ているページを収容しているページ枠が変更されると困る場合がある．この好例が，入出力データ域である．入出力オペレーションが行なわれている最中に，入出力データ域がページアウトされたとしたら，正しいデータの読み書きは全く保障できないことになってしまう．入出力域は，この理由のために入出力オペレーションが続行している間は固定される必要がある．

③ 特別な入出力装置のオペレーションは時間に強く依存している．たとえば，OCR とか MICR とかの装置は，非常に時間（タイミング）に依存している．これらの装置を対象とするプログラムはすべてページングをすることができない．もし，ページングが生じると，入力データがページング操作の途中で失なわれてしまうおそれがある．高度に時間に依存するプログラムのすべてのページは，その実行の間中固定する必要がある．

以上の条件にあてはまるページは，すべて長時間ないしは一定時間固定する必要がある．しかし，一般的にはページの固定化は極力回避すべきであろう．あるページが固定化されていると，そのページ枠はページングに利用できなくなってしまうからだ．

1） 監視プログラムの固定化

前述したように監視プログラムの中の中核プログラム部分は，実記憶装置の中に固定化され，一切ページングの対象とされない．中核プログラムの中のページ監視プログラムは，セグメント・テーブル，ページ・テーブル，ページ枠テーブルなどの作成と保守をおこない，ページ不在の割込みにもとづいてページアウト／ページインの操作を開始し，LRU 法にもとづいたページ置換の機能などを果してくれる．このページ監視プログラムがページアウトされたら，ページ置換のための面倒はみれなくなってしまう．また，他の中核プログラムの部分も要求時ページングによって実記憶装置へもってこられるとしたら，その間はシステムが待ち状態になり処理効率が一段とおちてしまいかねない．したがって，これらのページは，システムの操作中ずっと固定したままにしてお

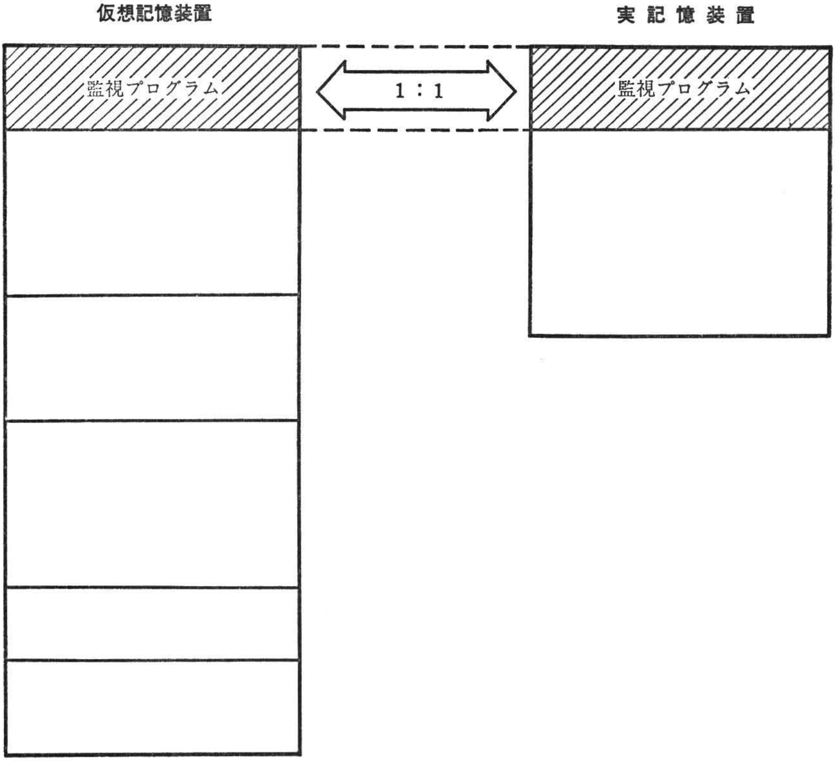


図 9-46 監視プログラム（中核部分）のページ固定化

く必要がある．つまり，永久的にページの固定化をしておくことが必要となる．

図 9-46 に示しているように，監視プログラムは仮想記憶装置上の記憶域と1対1の対応で実記憶装置の下位アドレスのページ枠に永久的に固定されてしまう（ここでいう永久的とは，IPL 操作後つぎの IPL がおこなわれるまでをいう）．

2) ページの短期固定化

前述したように，データの入出力域を含んでいるようなページは，入出力オペレーションをおこなっている最中にそのページがページアウトされてしまっ

たら大変なことになる．したがって，入出力域に使われているページは，入出力オペレーションが開始されてから完了するまで固定化する必要がある．入出力オペレーションが完了したあとは，そのページ枠を解放し，通常のページングに使用できるようにすればよい．この種のページの固定化を，**ページの短期固定化**と呼んでいる（**図9-47**参照）．入出力域の短期固定化は，ユーザ・プログラムで面倒みる必要はない．入出力監視プログラムの責任のもとで一時的に固定化させればよい．一方，ユーザ・プログラムからページの固定化を要求する必要が生じるかも知れない．ある種のデータ域だとか制御テーブル域を一時的に固定しておいた方がよい場合がある．このような場合には，特別なマクロ

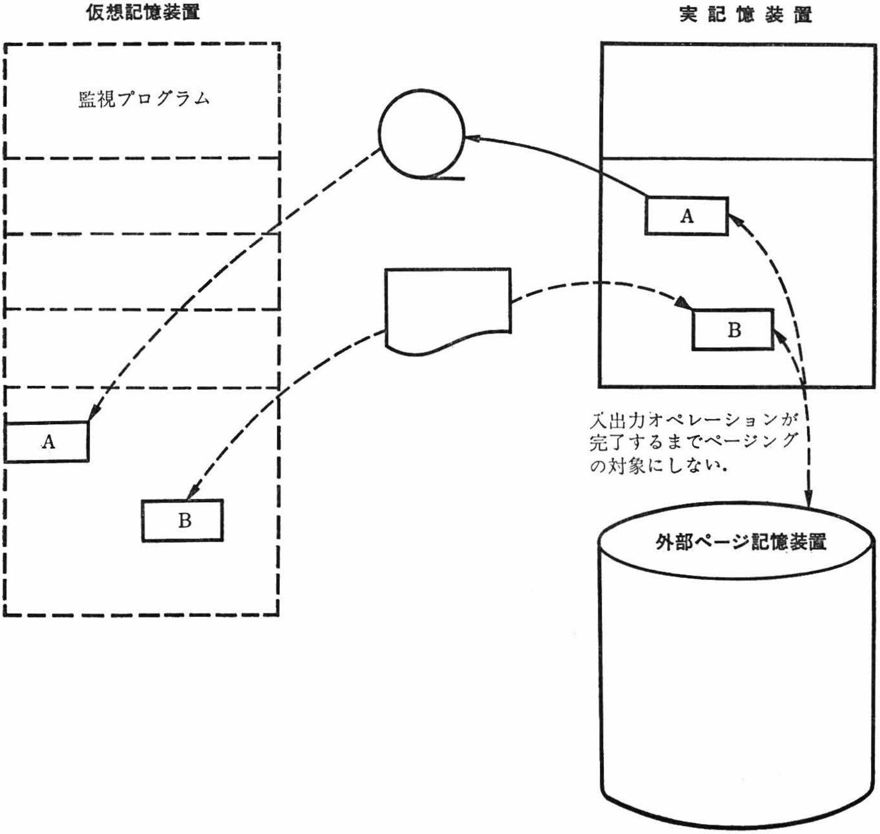


図 9-47　ページの短期固定化

命令（たとえば，PGFIX マクロ命令）で特定ページの短期固定化を要求できるようにしている．逆に，固定化したページをページング可能にするためのマクロ命令が用意されている（たとえば，PGFREE マクロ命令）．

3) ページの長期固定化

仮想記憶装置は，ページングを基準にして，**ページ不可能域**（non-pageable area）と**ページ可能域**（pageable area）との2つに分類することができる（**図 9-48** 参照）．ページ不可能域は，プログラムの実行中にページ不在が生じては困るプログラムの区域であり，監視プログラム域とV=R域からなる．V=R域は，監視プログラムの上位アドレスから始まり，V=Rラインまでの区域で

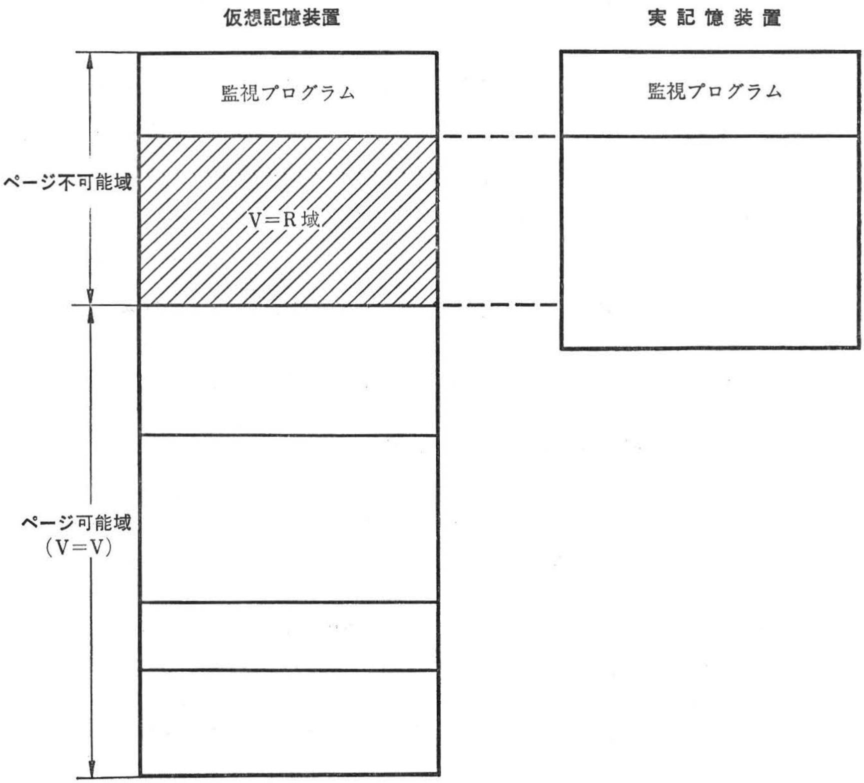


図 9-48　ページの長期固定化（V=R）域

ある．この区域は，実記憶装置の上限のアドレスを越えることは許されない．V=R ラインをどこまでにするかは，システムを生成するときに定義するのが普通である．

V=R 域の使用は，ジョブ制御カードで指定する．**図 9-49** でその例を示している．ADDRSPC パラメータで REAL と指定することにより，該当ジョブは V=R 域にローディングすることを要求している．さらに，REGION パラメータで領域の大きさを指示する．領域は V=R 域の上位アドレスから要求された大きさが確保される．ともあれ，V=R 域から確保された領域はすべて実記憶装置の対応する記憶域に固定されてしまう．つまり，該当ジョブの実行中はそのプログラムでは一切ページ不在が起らないことが保障される．このように，V=R 域を使ってジョブが実行される間ページを固定化してしまうやり方を，**ページの長期固定化**という．

V=R 域へプログラムがローディングされる場合には，プログラムは仮想記憶装置の V=R 域へ静的配置されながら，実記憶装置の連続したページ枠にロードされる．そして，V=R 域のページとそれがロードされたページ枠とは1対1の対応関係がある（**図 9-50** 参照）．換言するなら，このプログラムの仮想アドレスは，実アドレスと全く同じになる．ただし，両者のアドレスが同じであっても，動的アドレス変換機構は機能しアドレス変換のオペレーションは

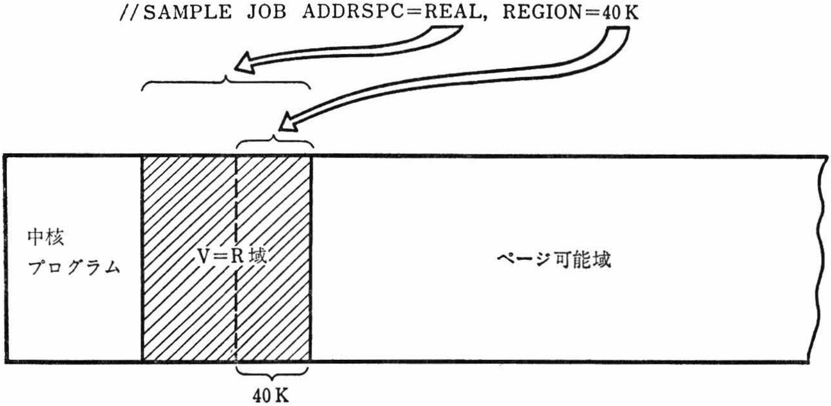


図 9-49　V=R域の指定

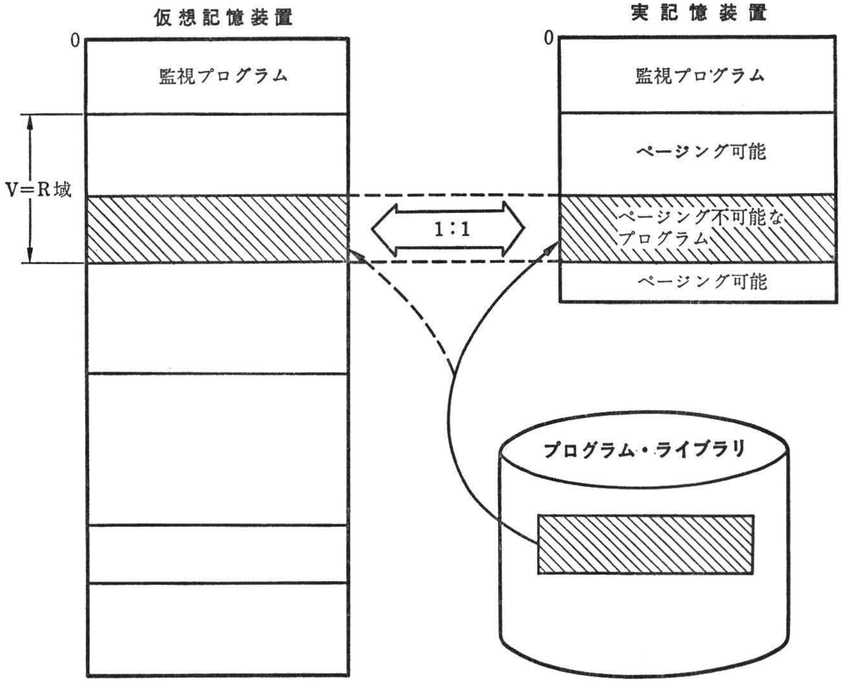


図 9-50　V=R 域へのプログラム・ローディング

おこなわれる．

V=R 域を利用する利点は，プログラムの実行中連続したページ枠を固定することにある．これ以外のページ枠は図で示しているようにページング可能なページ枠であり，他のプログラムのページングに適用できる．ともあれ，V=R 域でどんなプログラムでも実行させようとすることは得策ではない．V=R 域は，大きさが限られた区域であり，従来のシステムの主記憶装置の大きさとほとんど変らない．したがって，これを活用することは仮想記憶システムの利点を何ら生かしたことにならない．それにページ固定されたページ枠が多くなると，残りのページ枠でのページング率が急増し処理の効率が著るしく低下してしまうおそれがある．したがって，V=R 域は，次のように応答時間やタイミングに大きく依存するプログラムに対してだけ適用することを前提にしているのである：

- MICR や OCR のように時間に依存した処理を対象としたプログラム
- 実行中のチャネル・プログラムを動的に変更するプログラム
- 短い応答時間が要求されるプログラム（たとえば，オンライン・リアルタイム・システムのプログラムなど）

以上みてきたように，仮想記憶システムでは全てのページをページング対象にしているのではない．永久的に固定化するページ，プログラムの実行中固定化するページ，および特定のオペレーションの間だけ一時的に固定化するページがある．

8.　単一仮想記憶装置と多重仮想記憶装置

仮想記憶システムには，**単一仮想記憶装置**（single virtual storage）と**多重仮想記憶装置**（multiple virtual storage）とがある．これまで説明してきた仮想記憶システムは，単一仮想記憶装置を念頭においたものである．

単一仮想記憶装置は，その名の通りシステムには唯一個の仮想記憶装置しか存在しない（**図 9-51** 参照）．1つの仮想記憶装置をいくつかの区域に区分してそれぞれのユーザが利用するシステムである．この単一仮想記憶システムの特徴をまとめると次の4つに集約できる：

① 仮想記憶装置の大きさは，コンピュータ・ハードウェアのアドレス構造によって制約される．たとえば，24ビットのアドレス構造では最高16メガバイトの仮想記憶装置が許される．この値より小さい仮想記憶装置ならどんな大きさでもよい．ただし，実記憶装置より小さい仮想記憶装置は何ら意味をなさない．どの程度の大きさの仮想記憶装置が最適であるかを決定するのには諸々の要素が関係してくるが，一般には実記憶装置の4倍程度の大きさが目やすとなる．

② 単一仮想記憶装置は，1個のセグメント・テーブルによってマップされる（図 9-51 参照）．各々のセグメントに対応してそれぞれ1個のページ・テーブルが用意される．この件は，これまでに詳述してきた通りである．

③ 単一仮想記憶システムでは，仮想記憶装置はシステム全体のアドレス空

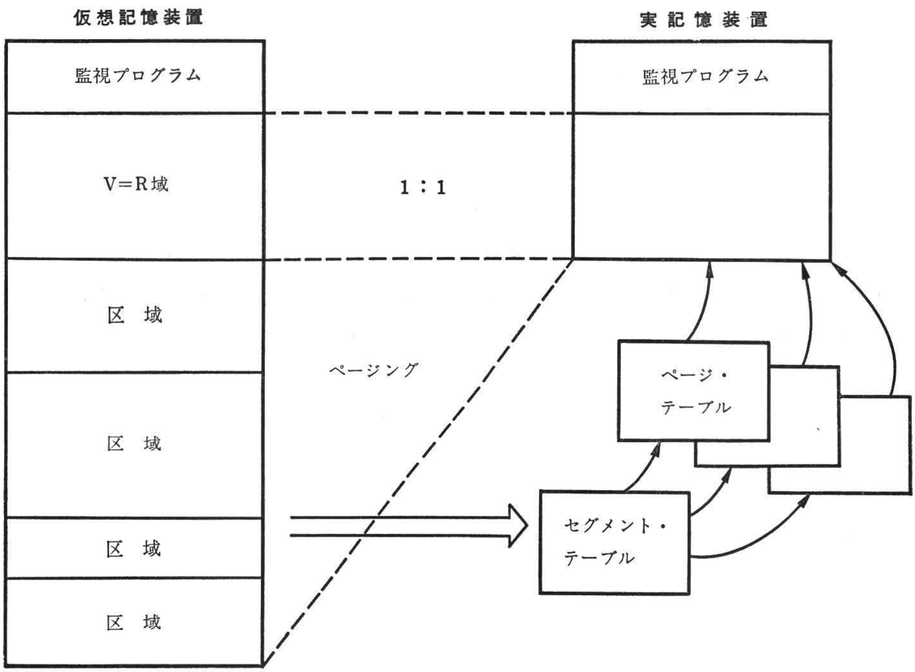


図 9-51　単一仮想記憶装置

間とみなされる．したがって，システムの中のプログラムは，システム制御プログラムを含めて1つの仮想記憶装置(つまり，1つのアドレス空間)にマップされる．それゆえに，1つのアドレス空間がいくつかの区域に分割されて個々のプログラムに割り当てられるので，それぞれのプログラムのアドレス空間はシステムのアドレス空間よりもかなり小さいものとなる．なお，仮想記憶装置のアドレス空間は，セグメント単位でユーザ・プログラムに割り当てられる．その結果，割り当てられた区域は何時でもセグメントの倍数の大きさとなる（ただし，V=R 域は除く）．

④ 単一仮想記憶システムでは，監視プログラムおよびユーザ・プログラムのすべてが同一仮想記憶装置上に入れられる．これは，従来のシステムが静的再配置でプログラムを主記憶装置にローディングしたやり方と基本的に変らない．仮想記憶装置を念頭にローディングするという点から差がで

るだけだ.

多重仮想記憶システムでは，単一仮想記憶システムと1つの点で大幅に異なる．単一仮想記憶システムでは，1つの仮想記憶装置をたくさんのユーザ（またはプログラム）で仲良く分けあって使うが，多重仮想記憶システムでは同時に働くプログラムがそれぞれ別個の自分専用の仮想記憶装置，つまりそれ自身のアドレス空間を用いる（**図9-52**参照）．単一仮想記憶システムでは1組のテーブル類を用いていたが，それぞれのアドレス空間に1組のテーブル類を用意して仮想記憶装置をマッピングする方法を採用することによって，多重記憶装置が実現できる．多重記憶装置の大きさは，単一仮想記憶装置の場合と同様に，コンピュータ・ハードウェアのアドレス構造によって制約される．24ビット・アドレスを使うシステムでは，それぞれの仮想記憶装置の上限はすべて16

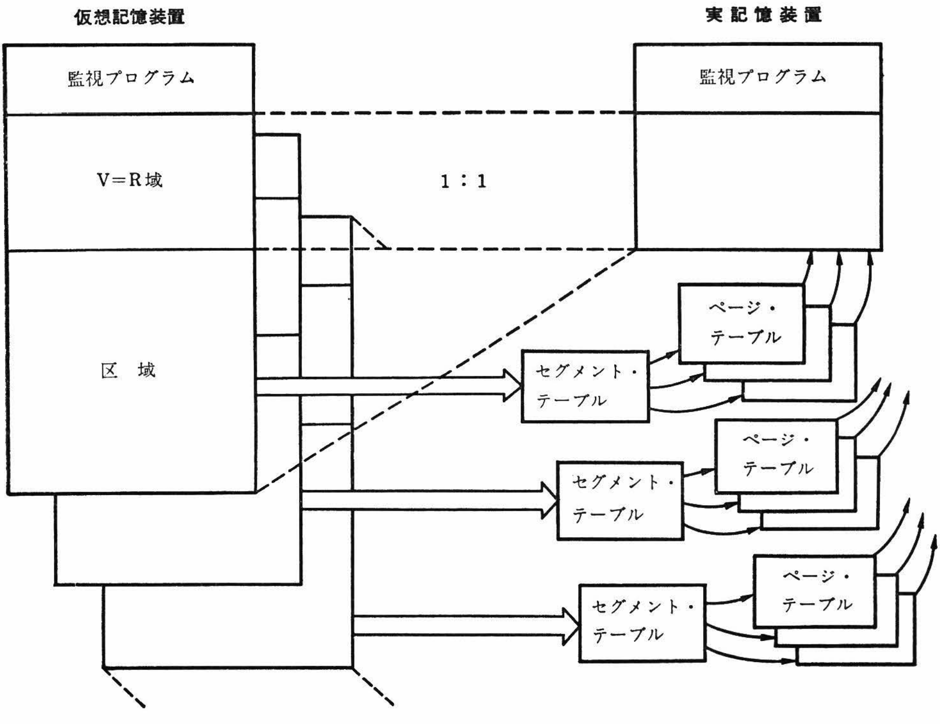


図 9-52　多重仮想記憶装置

メガバイトということになる．

単一仮想記憶装置は，1個の家屋をいくつかの部屋に分割して利用するようなものである．それぞれの住人には1個の部屋が与えられる．これが，プログラムのアドレス空間に相当する．1つの建物をそれぞれの住人用に間仕切ってしまうのであるから，個々の部屋は必然的に小さくなってしまう．従来のシステムのように，実記憶装置を分割して利用するのよりもはるかに大きなアドレス空間とはなるが，同時に処理するプログラムが多くなってゆくとたいしたアドレス空間でなくなってしまうおそれがある．

一方，多重仮想記憶装置は，それぞれの住人に1戸建ての家屋を与えるに等しい．台所（監視プログラム域）とか寝室（V=R 域）は必要としても，他のスペースは全く1人占めできる．全く間仕切りは不要だ．それだけ大きなアドレス空間がユーザに与えられる．同時に処理するプログラムがいかに多くなっても，それぞれに別個のアドレス空間が与えられるので何ら影響がない．つまり，多重仮想記憶システムの第1の魅力は，それぞれのユーザに大きなアドレス空間が与えられることだ．

1個の家屋をいくつかの部屋に分けたのではプライバシーは完全でない．他から侵害されたり，逆に他の部屋を侵害してしまうおそれがある．独立した家屋だと鎖さえしっかりしておけば，この恐れはほとんどなくなる．つまり，多重仮想記憶システムの第2のねらいは，記憶保護にある．それぞれの仮想記憶装置が独立しているので，他の区域を破壊したり，逆に破壊されることを防止できる．複雑で高度なシステムにおいて一段と信頼性を保障してゆこうというのが多重仮想記憶システムでもある．ただし，多重仮想記憶システムでは，ユーザ間での相互のコミュニケーションのむずかしさがある．隣りの部屋同志なら容易にやりとりができるが，独立した家屋だとそうはいかない．そこで，各々の仮想記憶装置上に**共通サービス域**（common service area）を設けて，相互の情報交換をおこなうようにしている．

多重仮想記憶システムにおいては，理論上仮想記憶装置の数に制限はない．しかし，実際的にはいくつかの制約が考えられる．実記憶装置の大きさ，外部ページ記憶装置に適用される補助記憶装置の大きさと転送速度，動的アドレス

変換のスピードなどが影響する．したがって，これらの要素を勘案して，最適な仮想記憶装置の数を設定する必要がでてくる．

三. 仮想記憶システムのためのハードウェア構造

第9.2節では，仮想記憶システムの概念的な構造を主体として，その仕組みを検討してきた．すでにお分りの通り，仮想記憶システムは単にソフトウエアの機能だけで対処しきれるものではない．動的アドレス変換機構をはじめとしたいくつかのハードウェア機構が用意される必要がある．

そこで，本節では仮想記憶システムを実現するためにどのようなハードウェア機能が追加されているかを説明してゆく．説明の都合上，IBM社のシステム／370をとりあげ，このシステムに追加されたハードウェア機能にもとづいて，その構造を明らかにしてゆくことにする．

1.　動的アドレス変換機構をサポートするプログラム状況ワード

プログラムの実行を正しくおこなうための情報が集約されている8バイトの特別なレジスタ（または記憶域）が，**プログラム状況ワード**（Program Status Word；略して **PSW**）であることはいうまでもない．この PSW は，従来のハードウェア・システムにも不可欠であった．割込みは，PSW が入れかわることによって引きおこされる現象である．ともあれ，PSW は仮想記憶システムをサポートするためにその情報の一部が従来の PSW とは異ったものになっている．その形式を示したのが，**図 9-53** である．この新たな PSW のもとでシステムが作動することを，特に**拡張制御モード**と呼んでいる．拡張制御モードであることを表示するために，ビット12が用いられ，ここに“1”ビットが立っていることによってシステムは拡張制御モードで作動する．

動的アドレス変換（Dynamic Address Translation；略して **DAT**）機構が働くこと表示するのには，ビット5が用いられる．このビットが“1”であると，システムは仮想アドレスを実アドレスに変換する機能を果たすのである．ビット0から7は，従来**システム・マスク**（system mask）が収容されて

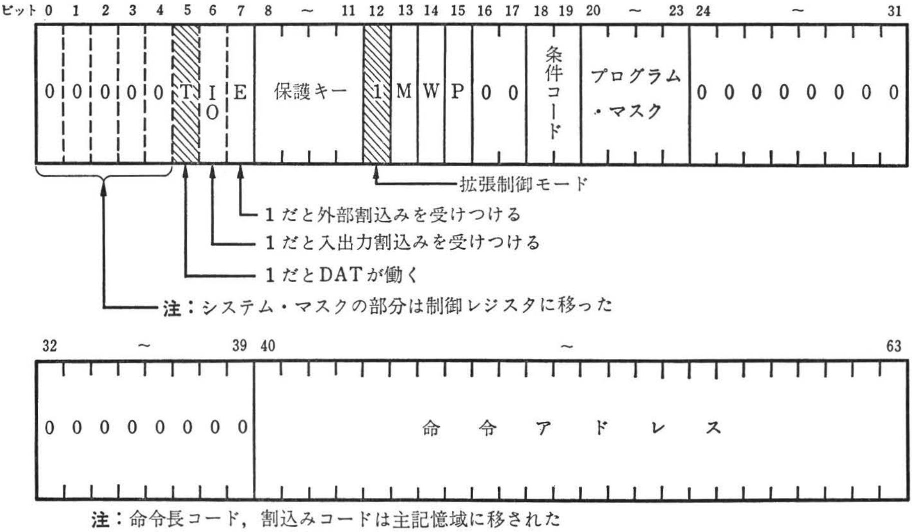


図 9-53 DAT をサポートする PSW の形式（拡張制御モードの場合）

いたが，拡張制御モードではこの部分は**制御レジスタ**（control register）に移されている．また，命令長コードと割込みコードは主記憶装置の特別な記憶域に貯えられる仕組みになっている．

システム／370は，仮想記憶システムではなく従来のシステムとして作動させることも可能だ．このためには，従来の形式の PSW を用いる（**図 9-54**）．従来の PSW の形式で作動させることを，拡張制御モードと区別するために，**基本制御モード**と呼んでいる．ハードウェアが基本モードで作動すべきか，それとも拡張制御モードで作動すべきかを判定するのには，前述のビット12を用いる．基本制御モードでは，このビットは必ず“０”である．この場合には，

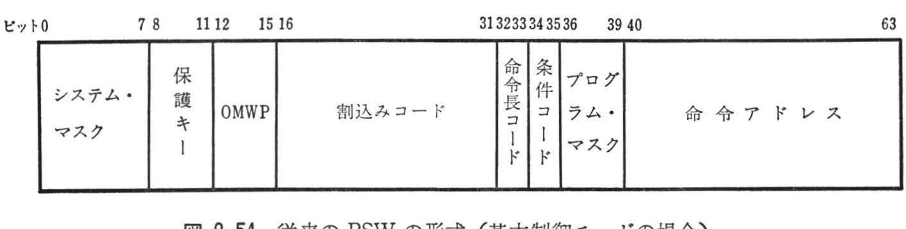


図 9-54 従来の PSW の形式（基本制御モードの場合）

PSWを**図9-54**の形式にみたててオペレーションをおこなう．逆に，ビット12が“1”であれば，拡張制御モードとみなしてオペレーションをおこなうのである．

ここで，制御レジスタについて補足しておこう，制御レジスタにはシステムを制御するための特別な情報が貯えられる．制御レジスタは複数個ある．さきにふれたシステム・マスクは制御レジスタ2に貯えられる．仮想記憶システムに適用するページ・サイズ（2Kバイトか8Kバイトか）およびセグメント・サイズ（目下システム／370には64Kバイトしか許されないが）などの情報は，制御レジスタ0に貯えられることになっている．また，セグメント・テーブル起点レジスタには，制御レジスタ1が適用される．

さて，PSWの命令アドレス（ビット40〜ビット63）に注目してみよう．仮想記憶システムの場合，ここに入る命令アドレスも仮想アドレスである．したがって，この命令アドレスは動的アドレス変換機構でのアドレス変換の対象となる．命令アドレス（仮想アドレス）が実アドレスに変換されたあとで，該当命令をフェッチしたり，該当命令にブランチしたりする．ところで，この命令アドレスは24ビット構成である（システム／370では，命令アドレスだけでなく，データ・アドレスも同じく24ビット・アドレスであるが）．したがって，仮想アドレスによるアドレス表現は，**図9-55**に示したようになる．2Kバイト単位のページと4Kバイト単位のページが許されているので，仮想アドレスは2種類に分かれる．2Kバイト単位だと変位部分が11ビット($2^{11}=2048$)ですむが，4Kバイト単位だと変位部分に12ビット（$2^{12}=4096$）必要である．1セグメントは64Kバイトに定められているので，2Kバイト・ページだと32ページ，4Kバイト・ページだと16ページ分から構成される．したがって，ページ番号の部分は，2Kバイト・ページでは5ビット（$2^5=32$），4Kバイト・ページでは4ビット（$2^4=16$）必要である．ともあれ，2Kバイト単位のページが適用されるか，それとも4Kバイト単位のページが適用されるかは，オペレーティング・システムの種類によってきまる．いずれにしろ，16,777,215（$2^{24}-1$）までの仮想アドレスの表現が許されるのであるから，最大256セグメント

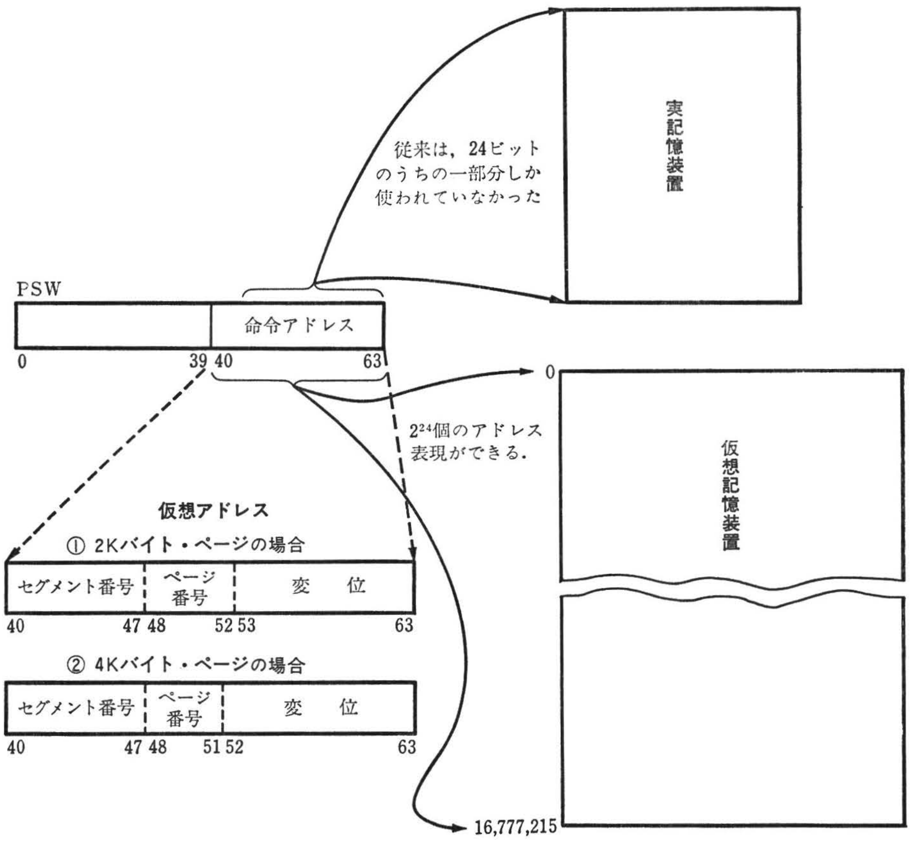


図 9-55　仮想アドレスによるアドレス表現

となる．したがって，セグメント番号の部分に8ビットとられるのである．

2.　動的アドレス変換機構の構造

システム／370の動的アドレス変換機構には2種類用意されている．セグメント・テーブルとページ・テーブルを用いた変換回路と**変換索引緩衝機構**(Translation Look-aside Buffer ; 略して**TLB**) とである．TLB は第9.2節でふれた「連想レジスターによる高速アドレス変換」に相当する．両者は同時に機能し，TLB によってアドレス変換が達成されると，テーブルを用いたアドレス変換は途中で中止されてしまう．一方，TLB によってアドレス変換が

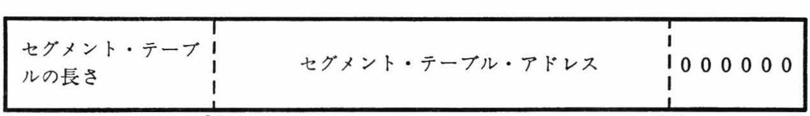
できない場合だと，テーブルを用いたアドレス変換が最後までおこなわれ，実アドレスが得られる．

1)　テーブルによるアドレス変換

テーブルによるアドレス変換では，前節で再三説明したようにセグメント・テーブルとページ・テーブルを用いる．これらテーブルも，ハードウェア機構ではない．いずれのテーブルも，制御プログラムが実記憶装置に作りあげ，維持してくれる性格のものである．しかし，動的アドレス変換機構でアクセスするテーブルであるという面からするとハードウェアと切り離すわけにはいかない．

主記憶装置にあるセグメント・テーブルは，**セグメント・テーブル起点レジスタ**（Segment Table Origin Register；略して**STOR**）でポイントされている．すでにふれたように，STORには制御レジスタ1が適用されている．STORの形式は**図9-56**の①で示している．8ビットから25ビット目にセグメント・テーブルのアドレスが入っている．ここに入っているアドレスは仮想アドレスではない．実アドレスである．また，ビット0から7にセグメント・テーブルの大きさが入っている．最大256セグメントまで許されるので本来セグメント・テーブルの記入項目は256個必要となる．仮想記憶装置がそれだけ大きくない場合には，セグメント・テーブルの記入項目は256個も必要ない．仮想記憶装置の大きさに見合ったセグメント・テーブルの記入項目数を用意すればよい．

①　セグメント・テーブル起点レジスタ



②　セグメント・テーブル記入項目

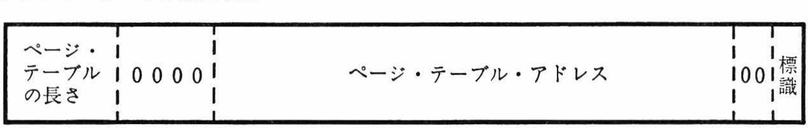


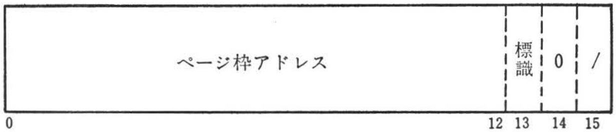
図9-56　セグメント・テーブル起点レジスタとセグメント・テーブル記入項目

ハードウェア・システムは，この長さフィールドでセグメント・テーブルの大きさを確認したうえで該当記入項目を捜し出す仕くみとなっている．

STOR にあるセグメント・テーブルの起点アドレスと仮想アドレスのセグメント番号とをもとにセグメント・テーブル上の所要の記入項目をみつけだす．その記入項目の形式は，**図 9-56** の②に示している．各記入項目とも 4 バイトである．ビット 8 から28にページ・テーブルのアドレスが収容される．31ビット目が無効ビットである．つまり，このビットは該当セグメントの参照が妥当なものであるかどうかを指示する．また，ビット 0 から 3 でページ・テーブルの大きさを表示する．これによって，ページ・テーブルは必ずしも 1 セグメント分のページ情報を収容していなくてもよくなる．セグメントの 1 部分しか用いていない場合には，その部分に対応するページだけの記入項目をもっておればよい．

一方，ページ・テーブルの記入項目の形式は**図 9-57** に示している．2Kバイト・ページの場合と4Kバイト・ページの 場合とでは，ページ枠アドレスのビット数が 1 つだけ異なる．ここに示されるページ枠アドレスは，24ビット・アドレスの上位12ビットないしは13ビットである．下位11ビットないしは12ビットは必ずゼロのはずであるので，記入項目のページ枠アドレスには示されない．標識で示した 1 ビットがページ不在を表示する無効ビットである．

① 2Kバイト・ページの場合



② 4Kバイト・ページの場合

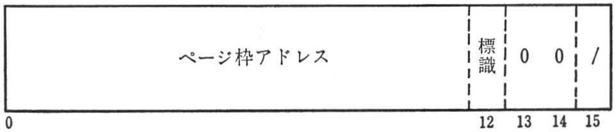


図 9-57　ページ・テーブル記入項目

さて，ここでテーブルによるアドレス変換回路（マイクロプログラミング）がおこなう変換過程を**図 9-58** でまとめておこう．

アドレス変換の必要が生じると，STOR の起点アドレスに仮想アドレスのセグメント番号を加算して，セグメント・テーブル内の該当記入項目をつきとめる．この場合，セグメント番号は2けた左にシフトしたうえで加算する．つまり，4倍してから加算する．これは，セグメント・テーブル内の各記入項目は4バイトであるから，4倍して加算することによって正しい記入項目を得ているのだ．ページ・テーブルが存在することが確認されると，ページ・テーブル・アドレスに仮想アドレスのページ番号を加える．この場合には左に1けただけシフトして，つまり2倍して加算することによって，ページ・テーブル内の該当記入項目のアドレスを算出している．ページ・テーブルの各記入項目は2バイトの長さだからだ．ともあれ，記入項目のアドレスが得られると，無効ビットをチェックして該当ページがページ枠に存在するかどうかを確認してから，実アドレスに変換する．実アドレスに変換するのには，ページ枠アドレスを上位の12ビットとして，そのうしろに仮想アドレスの変位12ビットをつけ加える．これによって，24ビットの実アドレスが得られる．

2） TLB によるアドレス変換

テーブルを用いた動的アドレス変換は，ハードウェア回路で遂行するので，高速ではあるが何回かにわたって主記憶装置をアクセスするので幾分の時間を要する．しかも，アドレス変換の必要のつどこのような変換がおこなわれるとしたら，1回1回の時間はわずかであっても，その累積時間は無視できないものとなってしまう．そこで，ほとんど時間を要しないでアドレス変換をおこなってしまう機構として，TLB 機構が併設されている．

TLB はきわめて高速の特殊レジスタであり，それを構成している各記入項目には現時点に最も近い時点で使用されたページの番号と，そのページがロードされているページ枠アドレスが保持される仕組みになっている．その構成を示したのが**図 9-59** である．TLB の記入項目の個数は，機種（モデル）によって異なる．図では8個の記入項目の場合を示しているが，16個のものもある

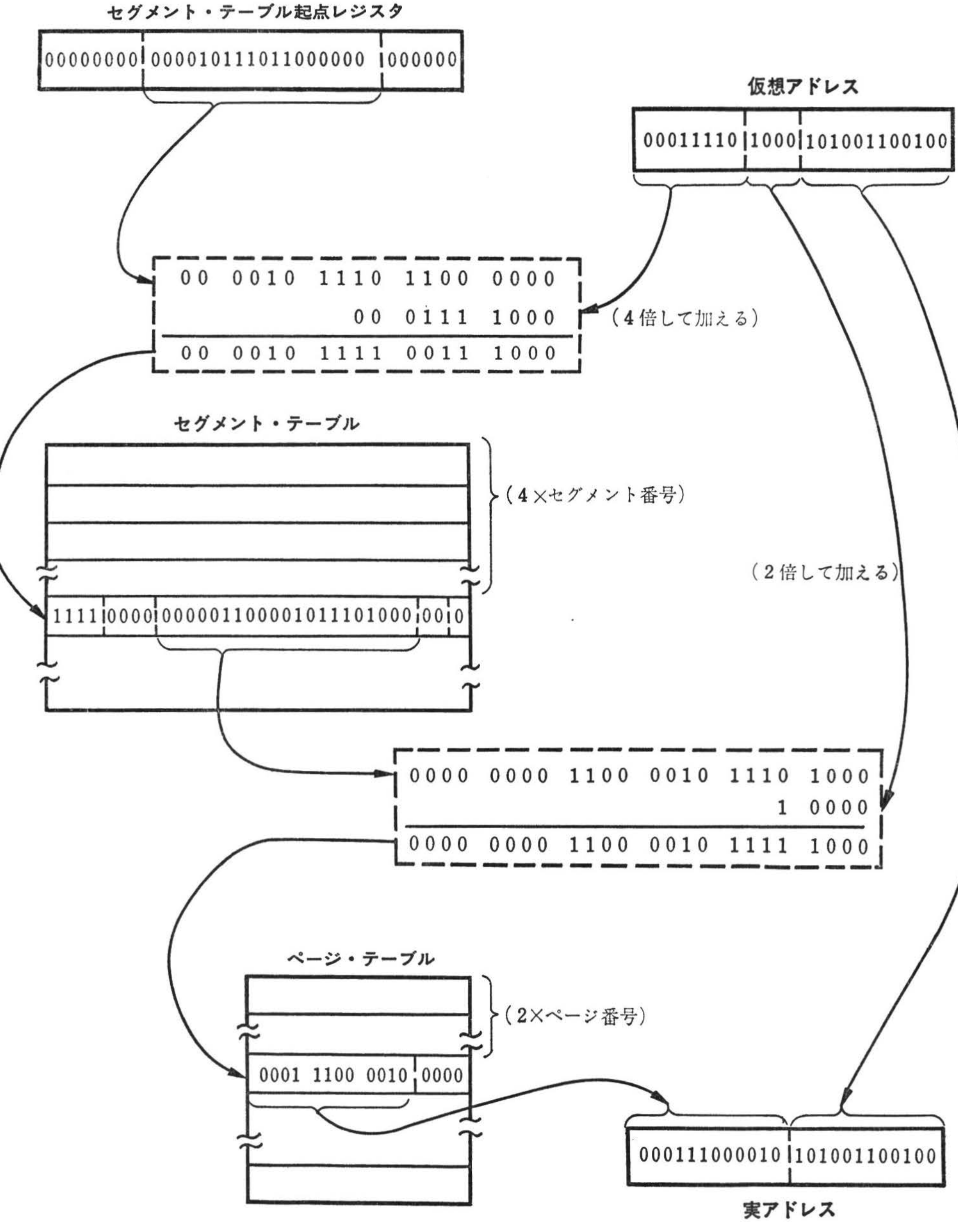


図 9-58 DAT 機構による実アドレスへの変換過程

セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス
セグメント番号	ページ番号	*	ページ枠アドレス

*：参照ビット

図 9-59 変換索引緩衝機構

し，大型機種だと128個から構成されている．

TLBの記入項目の置換の仕方もLRU法にもとづいている．そのために参照ビットが用いられる．参照ビットはそのページが参照されるとオンにされる．すべての記入項目の参照ビットがオンになると，はじめてオフにもどされる．TLBに登録されていないページが参照されると，そのページ番号（セグメント番号も含めて）が参照ビットが“0”の記入項目と入れかわり，ページ枠アドレスが更新される．このようにして，TLBにはいつでも最も近い時点に参照された8ページ（または，16ページないしは128ページ）分のページ枠アドレスが保持されているのである．

アドレス変換の必要が生じると，仮想アドレスのうちのセグメント番号とページ番号とが，TLBの各記入項目の中のセグメント番号およびページ番号と一挙に比較される（**図9-60**参照）．この操作は，テーブルを用いたアドレス変換と同時にスタートする．しかし，TLBでの操作の方がはるかに速い．比較の結果，TLBの記入項目の中に同じセグメント番号とページ番号がみつかると，そのページ枠アドレスに仮想アドレスの変位をつけ加えて実アドレスにするのである．同じセグメント番号とページ番号が発見されると，テーブルを使ったアドレス変換は自動的に中止されることはまえにもふれた．

このようなLRU法にもとづいたTLBを適用すると，ほとんどのアドレス変換はTLB機構で遂行され，一部のアドレス変換だけがテーブルによる動的

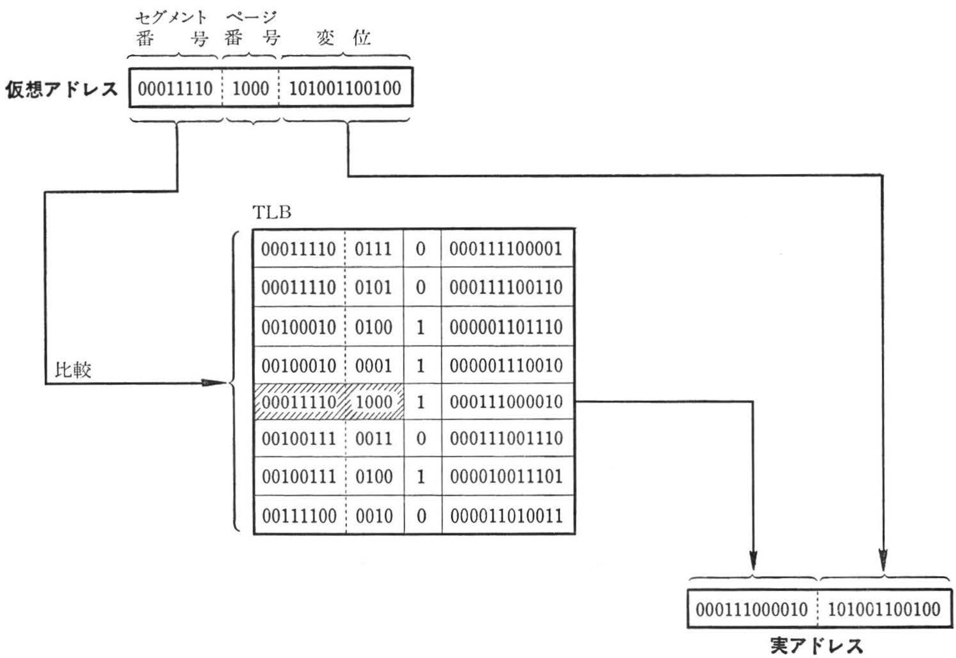


図 9-60 TLB 機構による実アドレスへの変換過程

アドレス変換にゆだねられる.

3. 記憶保護キー

システム/370 では, 2K バイトごとに**記憶保護キー** (storage protection key) を持っており, **図 9-61** のようなビット構成をしている. このうちの参照

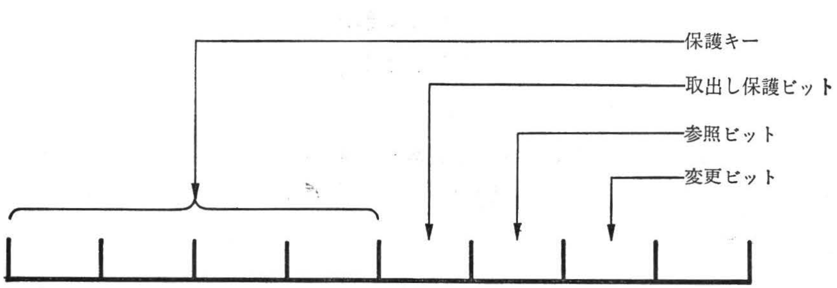


図 9-61 記憶保護キー

ビットと変更ビットの両方を利用して，ページ置換すべきページ枠を決定する．**参照ビット**はその記憶域が参照されたかどうかを示し，**変更ビット**はその記憶域内の内容が変更されたかどうかを示すものである．これらのビットをどのように使うかのアルゴリズムはオペレーティング・システムの種類によって異なる．ただし，もっとも参照の少なかったページとか，内容の変更がなされなかったページを優先してページ置換の対象にするという考え方は共通している．

これらのビットの状態は，ページ枠テーブルやTLBなどに反映され，ソフトウェアおよびハードウェア・システムが利用する．

4.　チャネル間接データ・アドレス指定機構

システム／370においては，実際の入出力オペレーションの面倒は入出力チャネルでみる．入出力チャネルは，チャネル・プログラムを実行することによって入出力オペレーションを遂行する．チャネル・プログラムは **CCW** (channel command word) の集まりである．ところで，入出力チャネルには動的アドレス変換機構が存在しない．したがって，入出力域のアドレスを実アドレスに変換するのには，ソフトウェアが介入せざるを得ない．この面倒は入出力監視プログラムがみる．入出力開始命令 (SIO) を出す前に，CCW の中のデータ・アドレス部分を実アドレスに変換する．そのあとで，入出力チャネルに制御権を渡す．ところで，入出力域が1ページ分におさまっていれば問題ないが，1ページを越えていたり，ページにまたがっていると単に CCW の中のデータ・アドレスを実アドレスに変えただけでは問題は解決しない．CCW の中のデータ・アドレスを実アドレスに変えることによって，最初のページにある入出力域に関しては正しく処理できても，残りの入出力域が存在するページが，実記憶装置のつぎのページ枠に連続して存在するという保障が全くないからだ．このような事態に対処するために設けられたハードウェア機構が**間接データ・アドレス指定機構** (Indirect Data Addressing feafure ; 略して **IDA**) である．

この機構は，入出力オペレーションに関しても動的な実記憶装置の使用を可能にする機構であり，入出力域が単一のページになくて，隣接しないページ枠

に分散していても処理できることを可能にする．**間接データ・アドレス指定**とは，CCW の中のデータ・アドレス部分で直接入出力域を指定するのではなく，別の場所で入出力域を指定することを意味する（**図 9-62** 参照）．別の場所で，各ページごとに入力域を指定したものを，**間接データ・アドレス指定リスト**（Indirect Data Addressing List；略して **IDAL**）という．図では，入出力域が 4 ページにわたっていると仮定して，4 つのデータ・アドレスを示している．この場合，データ・アドレス 1 では入出力域の最初のアドレスを指示し，他のデータ・アドレスでは各ページの最初のアドレスを指示する．IDA 機構を用いるためには，CCW のビット37に“1”をセットし，8 ビットから31ビットのデータ・アドレス部では，入出力域のアドレスの代りに IDAL のアドレスを指定しておく．こうしておけば，入出力チャネルは IDAL を用いて入出力オペレーションをおこなってくれる．なお，IDAL を作りあげたり，データ・アドレスを実アドレスに変換したりするのは，すべて入出力監視プログラムの仕事である．この種の仕事を一般に**チャネル・プログラムの変換**（channel program translation）と呼んでいる．

チャネル・プログラムの変換が所定の要領でおこなわれているならば，**図 9-63**に示しているように入出力域が複数ページにまたがり，実記憶装置内で分散

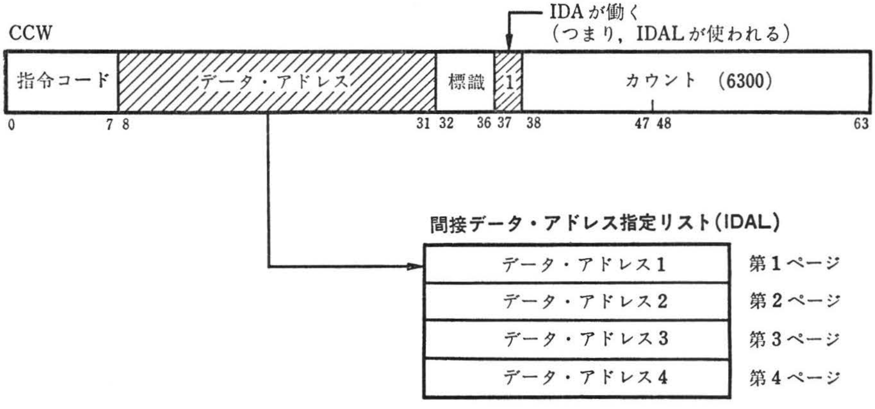


図 9-62　チャネル間データ・アドレス変換機構

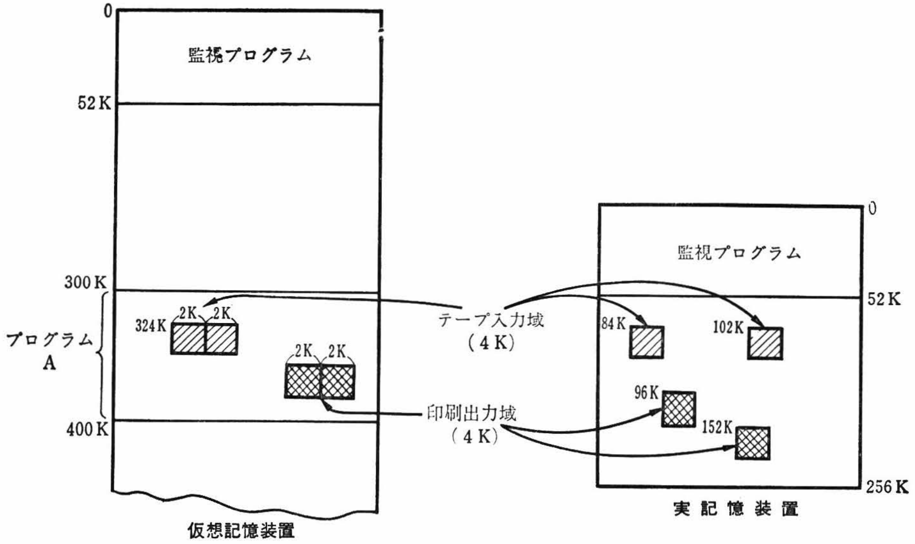


図 9-63　チャネル・プログラムの変換

したページ枠が与えられていても，IDA 機構の働きによって正しく入出力オペレーションが遂行されるのである．

仮想記憶システムを実現するために，以上のような拡張制御モード，制御レジスタ，動的アドレス変換機構，記憶保護キーの参照ビットと変更ビット，およびチャネル間接データ・アドレス指定機構がハードウェア機構として新たに付加されたのである．他のハードウェア機構は従来のものと本質的には変らない．

四．仮想記憶システムのためのソフトウェア構造

仮想記憶システムをサポートするオペレーティング・システムには，いくつかの種類がある．いずれのオペレーティング・システムでも「仮想記憶システムの概念的構造」の個所で詳述した仕組みを基本としている．ここでは，システム／370 の **OS/VS2** と呼ばれるオペレーティング・システムをとりあげて，仮想記憶システム用オペレーティング・システムの機能構造を紹介しよう．な

お，OS/VS2 には，単一仮想記憶システムの機能と多重仮想記憶システムの機能とがあるが，ここでは前者に焦点を合わせて説明してゆく．

仮想記憶システムをサポートする OS/VS2 といえども，前章までに解説してきたオペレーティング・システムを基礎としており，その構成もジョブ管理（ジョブの入力，ジョブの処理結果の出力，ジョブの実行準備，システムとオペレータとの連絡機能など），タスク管理（割込み処理，タスク・ディスパッチング，時間管理，主記憶域の管理，モジュールの管理機能など），データ管理（データの入出力のためのアクセス方式，入出力オペレーションの管理，入出力バッファの管理機能など）の三大機能を基本としている．これら機能に新たに仮想記憶システムをサポートするための**ページ監視プログラム**（paging supervisor）の機能が付加されている．したがって，本節ではページ監視プログラムを主体にその機能と構造を言及することになるが，ページ監視プログラムの説明に先立って，まずは OS/VS2 の仮想記憶システムの構造を明らかにしておこう．

1.　仮想記憶システムの構造

VS 2 では，1ページが4K バイト，1セグメントが64K バイトと定められている．したがって，1セグメントは16ページから構成される．仮想記憶装置の最大の大きさは16メガバイトであり，この場合には256セグメントになる．VS 2 の制御プログラムは，この16メガバイトの仮想記憶装置を従来のオペレーティング・システムが主記憶装置を管理したのと同様の方法で管理している．

したがって，ユーザは実記憶装置の大きさに関係なく，あたかも16メガバイトの主記憶装置を所有しているかのごとく，プログラム設計，コーディングあるいはオペレーションができる．

以下で OS/VS2 のもとでの仮想記憶装置の構造と構成，セグメント記憶保護，実記憶装置，外部ページ記憶装置と外部ページ・テーブルおよびプログラム・ローディングに関して概説する．

1）仮想記憶装置の構造

仮想記憶装置は，**非動的領域**（non-dynamic area）と**動的領域**（dynamic area）の2つの部分に分けられる（**図9-64**参照）．非動的領域は，制御プログラムのための領域であり，中核プログラムのための領域とページ可能監視プログラム域（SQAも含む）とに分割されている．一方，動的領域はジョブの実行のために**区域**（region）に割り当てることができる記憶域である．仮想記憶装置のこのような構成は，前章までにふれた従来のオペレーティング・システムの主記憶装置の構成と同じである．両者のちがいは，従来のオペレーティング・システムでは主記憶装置がこのような構成であるのに対して，VS2ではあくまで仮想記憶装置がこのような構成をとっていることだ．

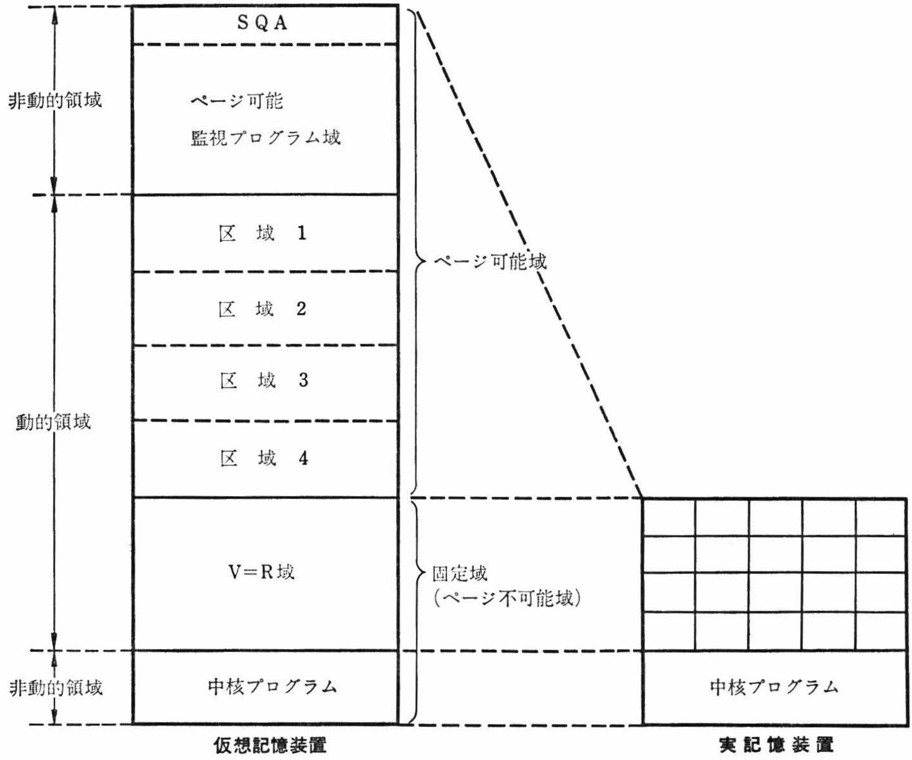


図9-64　仮想記憶装置と実記憶装置の関係

仮想記憶装置は，ページングを基準にしてその構造を把握することもできる（図 9-64 参照）．つまり，**ページング不可能域**（non-pageable area：**固定域**ともいう）と**ページ可能域**（pageable area）とである．ページ不可能域に割り当てられたページは実記憶装置内に固定され，ページングの対象とならない．ページ不可能域は，ページ単位の大きさで割り当てられる．ページ単位で割り振るのは，このページが実記憶装置内のページ枠内に固定できるからである．この結果，ページ枠を実際に必要とする数よりも多く固定しなくてすむ．一方，ページ可能域はセグメント単位に割り振られ，ページングの対象となる．つまり，ページ可能域にあるジョブは，仮想記憶装置をセグメント単位の大きさで確保するのである．ジョブの実行中，それらは要求時ページングの手法によって実記憶装置をページ単位に使用する．つまり，OS/VS2 は，仮想記憶装置の管理のためにはセグメンテーション手法を使用し，実記憶装置の管理のためにはページング手法を採用しているといえる．

ともあれ，仮想記憶装置は，要約すると次の4つの領域から構成されている：

① **非動的ページ不可能域**： これは，制御プログラムの**中核部分**（nucleus）によって使用される．中核プログラムは，入出力監視プログラムやページ監視プログラムなど監視プログラムのうち常に実証装置にとどまっていなければならない性質のプログラムによって構成される．

② **非動的ページ可能域**： この記憶域は，ページ可能監視プログラムによって使用される．監視プログラムは固定部分とページ可能部分とに分かれるが，ページ可能部分はそれほど使用頻度が高くないもの，あるいは重要度が低いものによって構成される．なお，従来のオペレーティング・システムでは，制御プログラムのうちの**再入可能**（reentrant）なモジュールのいくつかを**連係パック域**（Link Pack Area；略して **LPA**）に常駐させることができたが，OS/VS 2 ではこの種のモジュールはすべて非動的ページ可能域におかれ，仮想記憶装置上では常駐とされる．

③ **動的固定域**： この部分は，特別なユーザ・プログラムで使用される．こ

の区域は **V=R 域**といわれる．実行中にページ不在が起きては困るプログラムを実行するさい，ジョブ制御カードでその旨指示することによって割り当てられた区域は，すべて実記憶装置上に長期固定化される．

④ **動的ページ可能域**：この区域は通常のユーザ・プログラムで使用される．

2）システム構成要素の機能と構造

図 9-65 は仮想記憶装置上でのシステム構成要素の位置を示している．中核プログラムは，いうまでもなく，CPU や実記憶装置など重要なシステム資源の利用の仕方をコントロールしている．したがって，ページ監視プログラムも中核プログラムの一部分である．ともあれ，以下でシステム構成要素の機能と

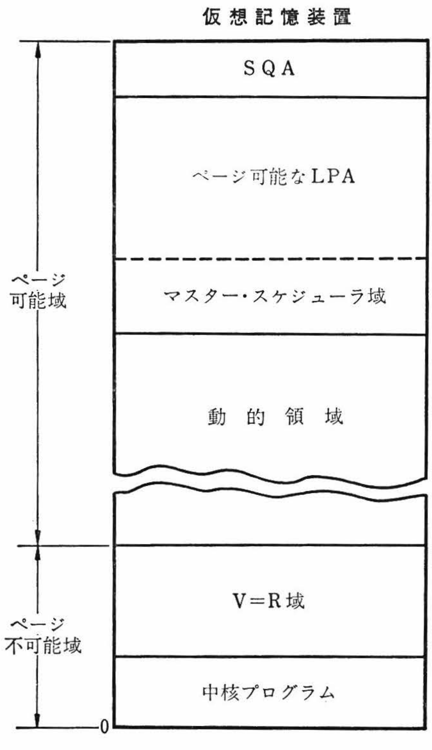


図 9-65　仮想記憶装置の構成要素

構造について解説しておこう．

① V=R 域

これは，第二節の「ページの固定化」の個所で説明したものと全く同じである．V=R の機能は，ジョブ・ステップを実記憶装置の連続した記憶域で実行させることにある．ちょうど，従来のシステムでプログラムを実行するのと同じ思想である．あるジョブが V=R 域で実行される場合，そのジョブで使用されるすべてのページは実記憶装置内に長時間固定される．プログラムが実行されている間中，ページングは行なわれず，チャネル・プログラムの変換もおこなわれない．

V=R 域は，複数個のジョブに割り振ることもできる．この区域の大きさはシステム生成時に指定するが，IPL 時にその大きさを変更することも可能となっている．なお，V=R 域の大きさは，実記憶装置の大きさを越えることはできない．

② システム待合せ域

システム待合せ域（System Queue Area；略して **SQA**）は，プログラムの実行中にシステムが参照する各種待行列や制御ブロックを格納するのに使用する．動的アドレス変換機構が用いるセグメント・テーブルも SQA 内に収容される．SQA に対しては，最少1セグメント割り振る必要がある．システムの必要度に応じてセグメント単位で増やしてゆくこともできる．システムの作動中，SQA に対して実記憶装置をページ枠単位で動的に割り当てる．なお，SQA の中の活動ページは，ページ枠内に長期固定される仕組みになっている．

③ ページ可能な連係パック域

先述したように，従来 LPA に格納していた再入可能モジュールはこの領域に常駐させることができる．また，従来のシステムでは**一時域**（transient area）に SVC ルーチンだとか入出力エラー回復ルーチンを呼び込んでいたが，VS2 では一時域がとられないので，これらのルーチンもページ可能な LPA に貯えられる．LPA 内に収容された各種ルーチンは，共用可能なルーチンであり，ページングされる．

従来のオペレーティング・システムでの LPAの 確保にあたっては，常駐させる必要のあるルーチンの大きさに見合った主記憶域をあらかじめ定義しておく必要があった．数10Kバイトから，場合によっては100K バイトを越える主記憶域が LPA 用に確保された． ところが，LPA が使用されないときは，この記憶域は無駄になってしまう．したがって，極力 LPA 用の記憶域を小さくしようと試みられた． OS/VS2 では， すべての常駐ルーチンに必要な大きさの仮想記憶域をページ可能 LPA に与えるが，これらのルーチンが使用されないときは，実記憶装置は割り当てられない．使用されるときだけ要求時ページングによって，実記憶装置内のページ枠に移る．それゆえに，すべての常駐ルーチンを収容するに足る大きさの仮想記憶装置を LPA 用に割り当てたとしてもあまり問題はないのである．なお，ページ可能な LPA は最小15セグメント(960K）の仮想記憶装置を必要とするとされている．アクセス方式ルーチンのような選択機能を常駐させると，LPA はさらに大きなものとなる．

④ マスター・スケジューラ域

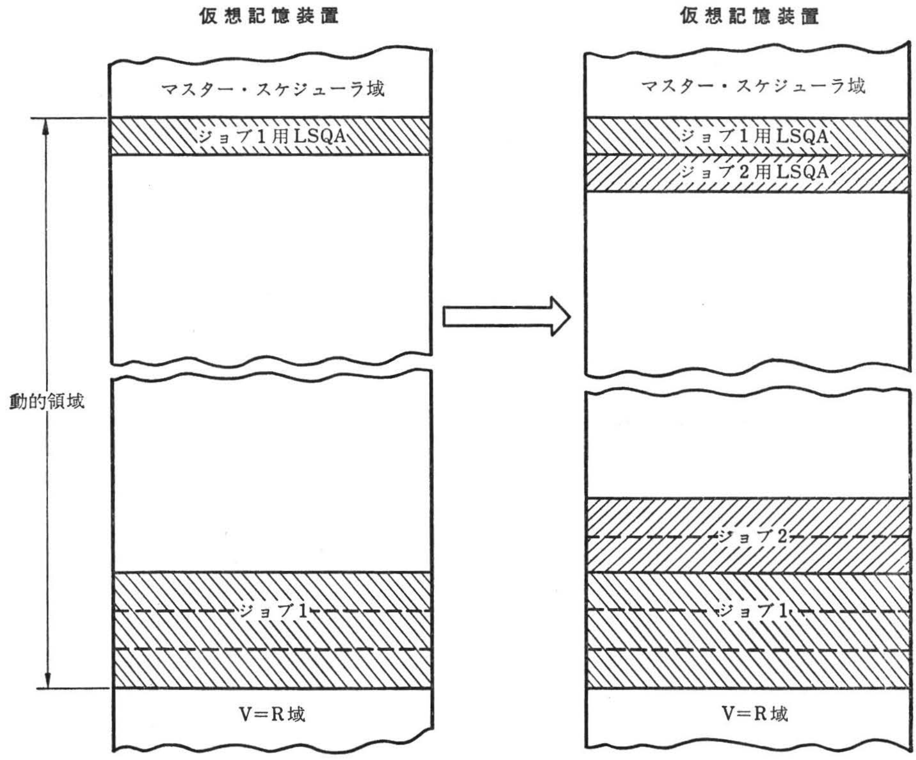
マスター・スケジューラは，いうまでもなくオペレーティング・システムとオペレータとの間でのコミュニケーション機能を果たす．このマスター・スケジューラを収容する区域が，マスター・スケジューラ域である．マスター・スケジューラ域としては，最低2セグメント必要とする．仮想記憶装置上のページ可能域にある他のシステム構成要素と同じように，マスター・スケジューラも必要に応じてページングされ，実記憶装置内のページ枠に移される．

⑤ 動的領域

システムが，各構成要素に対して仮想記憶装置を割り振ると，あとに残るのは動的領域に属する部分である．この動的領域を分割してできた区域の中で，ユーザ・ジョブが実行される．仮想記憶装置は，ジョブに対してセグメント単位で割り振られる．つまり，あるジョブに対して100K 必要な場合には，仮想記憶装置のうちの2セグメントからなる区域が割り当てられる．さらに，1セグメントが**ローカル・システム待合せ域** (Local System Queue Area；略して **LSQA**）として，動的領域の上位アドレスから割り当てられる．

LSQA は，システムが SQA を使っているのと同じように，ジョブに関係する各種制御ブロックや待行列を収容するために用いる．これは，ジョブの実行中にシステムが参照し，必要に応じて更新する．なお，ジョブに割り当てられた区域に関するページ・テーブルも LSQA に格納されている．区域ごとに（つまり，ジョブごとに）専用の LSQA をもうけると，ジョブのスケジューリングと実行段階に区域に対しての独立性を保障しうる利点がある．

ここで，動的領域がどのように割り当てられてゆくかを考察しておこう．ジョブ1と名づけられるジョブを仮定しよう．このジョブを実行するのに 180K バイトの仮想記憶装置を必要とするものとする．この場合，ジョブ1のプログラムの実行のために3セグメント（192K バイト），LSQA 用として1セグメ



1)「ジョブ1」の開始　　2)「ジョブ2」の開始

9-66　動的領域の割り当て (1)

ントが必要となる．これらがどのように割り当てられるかを，**図 9-66** の左側で示している．区域は動的領域の下位アドレスから確保されてゆき，LSQA は逆に上位アドレスから確保されてゆく．

ここで，もう1つのジョブ（ジョブ2）が開始されるとしよう．このジョブを実行するのに100K の記憶域を必要とするものとする．この場合，仮想記憶装置は，図 9-66 の右側で示しているように割り当てられる．2つのセグメントがプログラムの実行のために，別の1セグメントが LSQA 用にとられる．

このように，動的領域はジョブ用の区域と LSQA 用の区域にどんどん割り振られてゆくのである．ところで，ジョブが終了した場合はどうなるであろうか．いま，ジョブ1が終了し，システムはジョブ3をスケジュールするものと仮定しよう．この場合，ジョブ3はプログラムの実行のために100K バイトを必要とするものとする．この様子を示したのが **図 9-67** である．左側ではジョブ1が終了した時点での仮想記憶装置の状態を示している．ジョブ1が占めていた区域と LSQA 用のセグメントは解放される．右側では，ジョブ3が開始された様子を示しているが，プログラムの実行のために2セグメントが割り当てられ，LSQA のために1セグメントが割り当てられる．

OS/VS 2は，それぞれの区域に対して動的にセグメントを割り振る．ジョブが動的領域を使い果すまで，ジョブのスケジューリングをすることができる．また，ジョブが終了することによって解放されたセグメントは，別のジョブをスケジュールするために動的に割り当てることができるのである．

3) セグメント記憶保護

これまでみてきたように仮想記憶装置は，セグメント単位で管理されている．仮想記憶装置をセグメント単位でユーザに割り振っているので，記憶保護もセグメント単位でおこなう方法がとられている．

OS/VS2 では，プログラムを実行しているさいに，そのプログラムが収容されているセグメント，および共用可能なシステム・セグメント内にある仮想アドレスしか参照できない仕組みになっている．その他のセグメント内の仮想アドレスは一切参照できない．つまり，該当ジョブ用の区域と LSQA および共

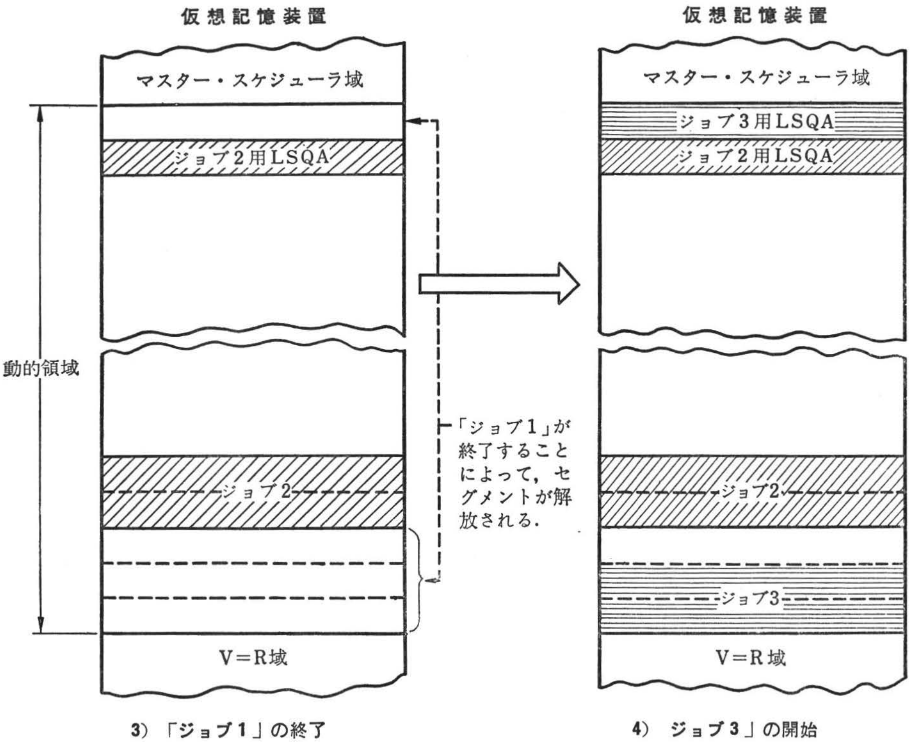


図 9-67　動的領域の割り当て　(2)

用可能システム・セグメント以外のセグメントは完全に記憶保護される．この機能を**セグメント記憶保護**と呼んでいる．この機能があるので，システムの信頼性と機密保護とが一段と強化されることになる．ユーザは，システムによって承認されないかぎり，他のユーザ・ジョブまたはシステム構成要素を不注意に，または意識的にアクセスしたり，変更したりすることは一切できないからである．

システムは，セグメント記憶保護をおこなうのに，セグメント・テーブルを利用している．セグメント・テーブルの各記入項目には標識ビットが設けられていたことを記憶されているだろう．この標識ビットは，別名「無効ビット」ともいう．この無効ビットがセグメント記憶保護のための重要な役割りを果たす．つまり，無効ビットは，そのセグメントに対する参照が妥当であるかどう

かを指示する，仮想アドレスで無効ビットが“0”であるセグメントを参照した場合には問題はない．正常に動的アドレス変換が遂行される．無効ビットが“1”だと，アドレス変換はその場で中止され，記憶保護侵害がおこなわれたことを知らせる．

このからくりを，例をもとに考えてみよう．いま，プログラム1と名づけられたプログラムを実行しようとしているものとしよう．このプログラムが仮想記憶装置のセグメント7からローディングされると仮定する．プログラム1が160Kバイトの記憶装置を必要とするなら，システムはこのプログラムのために3個のセグメント（セグメント7，8，9）を確保してくれる．また，LSQA用として1セグメントを確保する．プログラム1の実行を始めるとき，システムはセグメント7,8,9ならびにLSQA用セグメントに対応するセグメント・テーブルのそれぞれの記入項目に含まれている無効ビットを“0”にする．その他のすべてのセグメントに対応する記入項目の無効ビットは“1”にする．したがって，プログラム1がセグメント7，8，9ならびにLSQA用セグメント内の仮想アドレスを参照した場合には，無効ビットが“0”であるからアドレス変換はそのまま続行され，正しい実アドレスが得られる．逆に，これら以外のセグメント内の仮想アドレスを参照すると，無効ビットが“1”であるためにアドレス変換は中止され，アドレス参照は不可能となる．

プログラム1が割込みを受け，別のプログラムに制御権が移る場合はどうなるであろう．この場合には，プログラム1に対する記入項目の無効ビットをオンにし，実行が再開されたプログラムに対応する記入項目の無効ビットをオフにしてくれる．そのあとで，新しいプログラムの実行をおこなうので，やはりセグメント記憶保護は保障される．

セグメント記憶保護を完全にコントロールしてゆくために，OS/VS2は2個のセグメント・テーブルを使用している．1つは監視プログラム状態で用いるセグメント・テーブルであり，他は問題プログラム状態で使用するセグメント・テーブルである．システムが監視プログラム状態であるときは，仮想アドレスの動的変換のために第1のセグメント・テーブルを使用する．このテーブル内のすべての記入項目の無効ビットは“0”の状態にされている．したがって，

監視プログラムは，仮想記憶装置上のどのセグメントでも参照できる．一方，問題プログラム状態では，仮想アドレスを変換するために第2のセグメント・テーブルを使用する．このセグメント・テーブルでは，実行中の問題プログラムに関連するセグメントと共用システム・セグメントの記入項目の無効ビットだけが“0”の状態にされたうえで使用される．先に説明したプログラム1の例は，このような状態で実行される場合である．問題プログラム状態で実行されるプログラムは，そのプログラム専用のセグメントと LPA の中にある共用システム・セグメントだけしか参照できないことになる．これによって，他のセグメントは完全に保護されてしまう．

以上のようなセグメント記憶保護の機能によって，システムとユーザ・プログラムに対する高度な記憶保護が実現できるのである．

4） 実記憶装置の構造

図 9-68 は，仮想記憶装置を実記憶装置にマッピングした状態を示している．仮想記憶装置上のページ可能域のシステム・ルーチンおよびユーザ・プログラムで実記憶装置上のページ枠が共用される．ここで注意しなければならないのは，システムの中核プログラムが実記憶装置上に固定されているということである．これによって，システムおよびユーザ・ジョブに対するサービスが迅速化する．この点に関しては，これまで再三指摘してきた通りである．

ジョブ・ステップが V=R 域で実行されている場合には，そのすべてのページ群が4K単位で割り当てられ，ジョブの実行の間中ページ枠に固定される．しかも，このページ枠はお互に連続したものであることを前提にしている．**図 9-69** で V=R ジョブの実行の様子を示している．V=R のジョブ・ステップのページは，仮想アドレスに対応する実記憶装置アドレス内に1対1の対応で固定される．このことを図 9-69 では，仮想記憶装置と実記憶装置上の斜線で表現している．ジョブがたとえ V=R 域で処理されるものであっても，やはり LSQA 用の仮想記憶セグメントを1個もっている．LSQA の活動ページに関しては，図 9-69 でページ枠上に斜線をほどこした小さな長方形の囲みで表わしている．

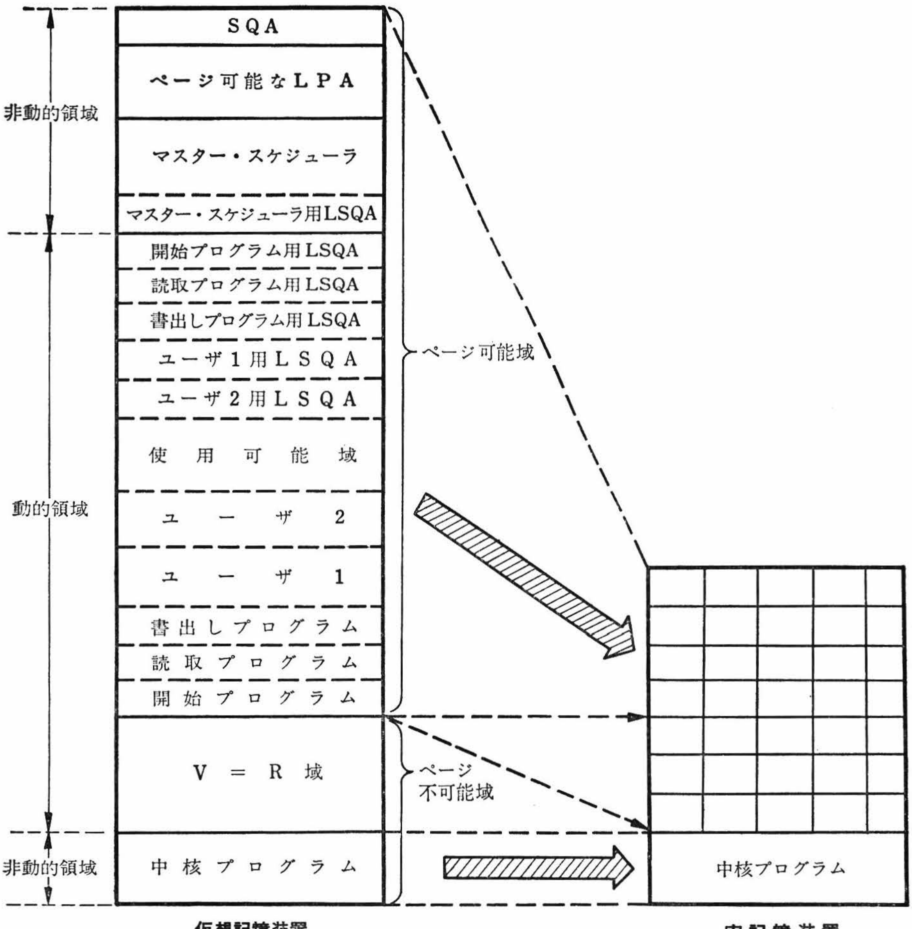


図 9-68　仮想記憶装置と実記憶装置

一般に V=R ジョブは，動的領域にスケジュールされて実行が開始され，その実行が終了されないと困る．たとえ，V=R のジョブ・ステップが，その実行の間中，すべてのページ枠を固定したにしても，残りの実記憶装置ページ枠は，要求時ページングを通じてページ可能仮想記憶装置上の別の活動ジョブによって利用される運命にある．

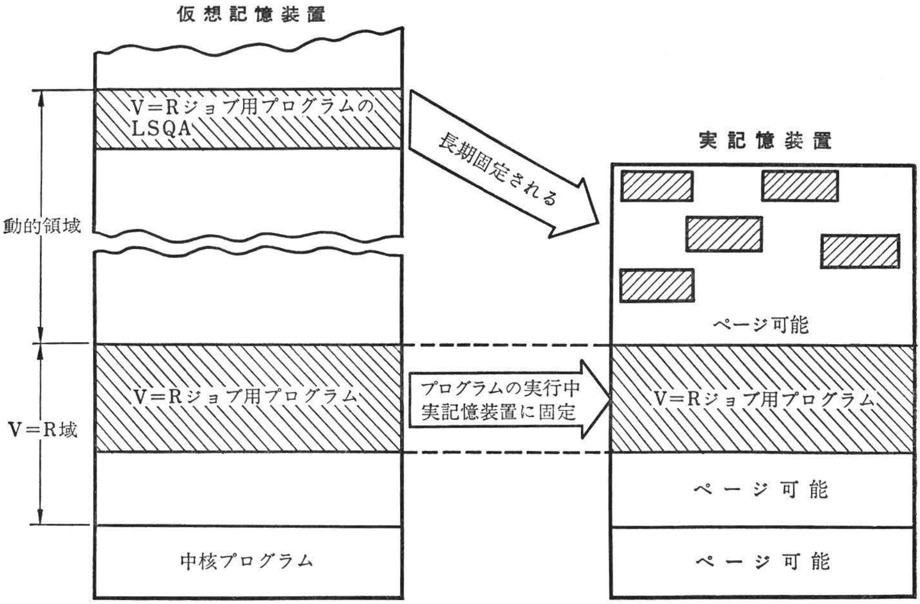


図 9-69 V=R ジョブの実行

5) 外部ページ記憶装置と外部ページ・テーブル

すでに何回となく説明してきたように，必要とするページが実記憶装置上のページ枠に収容しきれない場合には，外部ページ記憶装置上のスロットに貯える．では，一体外部ページ記憶装置上のどのスロットに貯えるのであろうか．このための判断に使用されるのが，実記憶装置上にある外部ページ・テーブルである．

仮想記憶装置は，外部ページ記憶装置と1対1でマッピングされるのではない．外部ページ・テーブルを使って，外部ページ記憶装置上の適当なスロット場所をマッピングする方法がとられている．

これまで，ページングとは，実記憶装置と外部ページ記憶装置との間でのページの転送のことであると定義してきた．ページインのときには，あるページが外部ページ記憶装置から実記憶装置に移されることになり，逆にページアウトのときは，ページが実記憶装置から外部ページ記憶装置の方へ移されることである．ところで，仮想記憶装置上のある部分がページングの対象にならない

としたら，そのページを格納するためのスロットは必要ないことになる．すでにふれたように，仮想記憶装置上のページ不可能域の部分はページングされない．中核プログラムは，オペレーション中はいつでも実記憶装置に固定されたままである．ジョブ・ステップを V=R として実行しているときには，ジョブの実行中，このステップのページ群は実記憶装置内に固定されている．また，SQA セグメントに割り当てられているページや LSQA に割り当てられている一部のページは，やはり長期的に固定される．これらのページ部分は，ページングされないので，これらの内容を保持しておくためのスロットは不要となる．

では，ページングされるシステム構成要素はどうであろうか．システム構成要素である LPA やマスター・スケジューラはページングされる．これらの構成要素は必然的に外部ページ記憶装置を利用する．しかし，動的領域のうちまだ未割当てのセグメントは，外部ページ記憶装置は利用しない．同様に，セグメント内の未使用のページも外部ページ記憶装置を使用しない．このような未割当てのセグメントやページは，システムにとっての潜在的なアドレス空間である．

図 9-70 の左側ではページング可能な仮想記憶装置部分を表示している．ジョブ1とジョブ2とが，動的領域内の区域を使用して実行されている．この場合未割当ての 222 個のセグメントは，外部ページ記憶装置のどの部分も使用していない．この部分は，未使用の仮想記憶装置ないしは潜在的なアドレス空間を示す．4個のセグメントを必要とする新たなジョブがスケジュールされたら，未使用の動的領域が割り振られ，潜在的な仮想記憶装置は 218 セグメントに減ることになる．

ページングがおこなわれるときには，ジョブのために確保されたセグメントが外部ページ記憶装置を使用することになる．図 9-70 でジョブ2用プログラムの二番目のセグメントに注目して欲しい．この例では，ジョブ2が 112K バイト必要とするものと仮定しているので，2番目のセグメントの4ページ分（16K バイト）はジョブの実行の最中まったく使用されない．したがって，この4ページ分に関しては，ジョブの実行中外部ページ記憶装置は全く不要であ

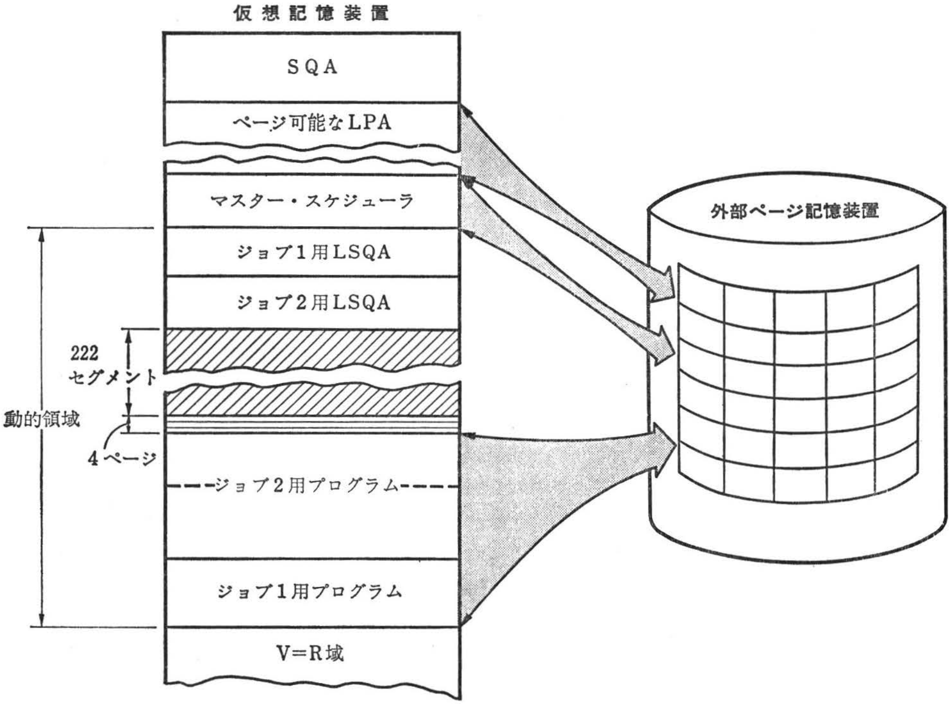


図 9-70 仮想記憶装置と外部ページ記憶装置の関係

る.

OS/VS2 では, 外部ページ記憶装置を動的に使用する. ユーザは仮想記憶装置をバックアップするに十分な外部ページ記憶装置のスペースを確保しておかなければならないが, さきにみてきたように実際に使用される外部ページ記憶装置の空間は, 仮想記憶装置の空間よりもかなり少なくてすむ.

ここで, 外部ページ・テーブルに関して若干補足しておこう. ページ・テーブルが実記憶装置内のページ枠内に存在するページの場所をマッピングするのに対して, 外部ページ・テーブルは外部ページ記憶装置上のスロットに収容されているページの場所をマッピングする. すでに説明してきたように, ページ・テーブルごとに, それに対応する外部ページ・テーブルが用意される. 仮想アドレスを変換するさいにページ不在が生じると, システムは対応する外部ページ・テーブルを用いて, 外部ページ記憶装置上の該当スロットを捜し出し,

ページインする．もし，ページ置換がおこり，しかも置換対象のページ内容がすでに変更されている場合だと，そのページをページアウトする必要がある．この場合，システムは実記憶装置から外部ページ記憶装置にページを移す時間がもっともすくなくてすむスロットを選定する．ページアウトが生じると，システムは，ページ・テーブルの該当記入項目の無効ビットをオンにし，外部ページ・テーブルの該当記入項目に，ページアウトされたスロットの場所（アドレス）を入れる．

OS/VS2は，動的アドレス変換機構と各種のテーブルを使用して，仮想記憶システムを実現している．各種記憶装置にテーブル類がどのように関連しているかをまとめたのが**図9-71**である．セグメント・テーブルとページ・テーブルは動的アドレス変換に使われる．セグメント・テーブルはアドレス変換のほかに，仮想記憶装置の割り振りや記憶保護の管理にも使用されている．ページ・テーブルとそれに対応する外部ページ・テーブルは，実記憶装置と外部ペー

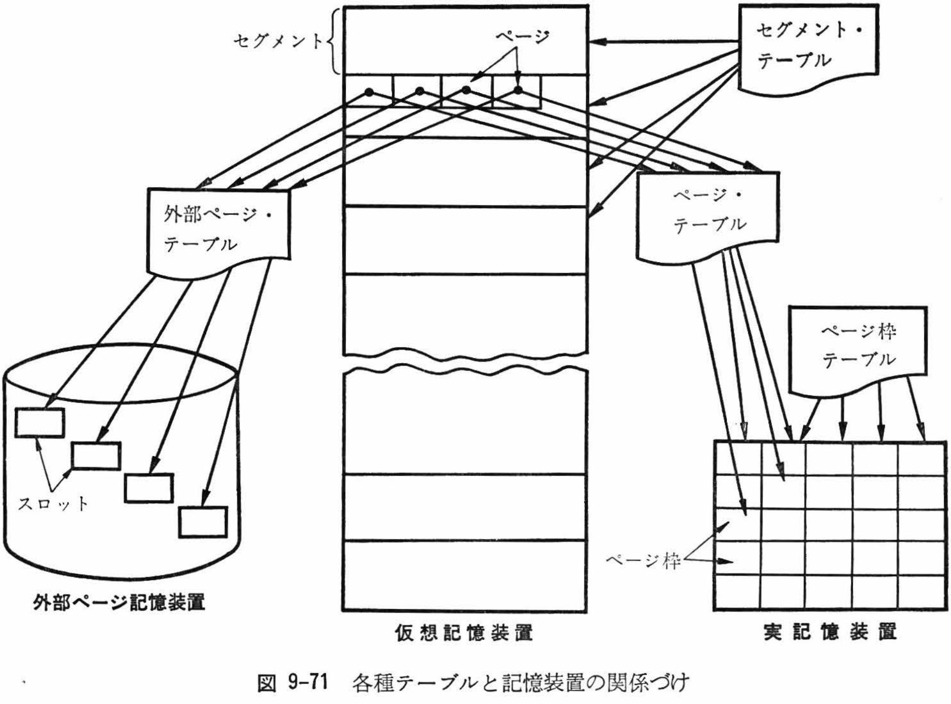


図 9-71　各種テーブルと記憶装置の関係づけ

ジ記憶装置との間でのページ転送を制御している．また，ページ枠テーブルは，実記憶装置の割当て管理に用いられる．

これらのテーブル類と動的アドレス変換機構によって，仮想記憶システムが実現されているのである．

6） プログラムのローディング

あるジョブがスケジュールされると，最初のジョブ・ステップに関するプログラムが，仮想記憶装置上に確保された区域にローディングされる．OS/VS2のプログラム・ローディングは，従来のシステムのプログラム・ローディングと非常によく似かよっている．唯一の相違点は，OS/VS2ではプログラムが仮想記憶装置上にローディングされるということである．

仮想記憶システムにおいても，従来のシステムと同じように，プログラムはプログラム・ライブラリに貯えられている．ここでプログラム1を仮想記憶装置上の区域にローディングしようとしている状態を仮定しよう（**図9-72**参照）．

プログラム1は，OS/VS2のシステム・ライブラリに貯えられている．これは，コンパイルまたはアセンブルされたあと，連係編集されたうえで相対アドレスをもったロード・モジュールの状態である．したがって，プログラム1の起点のアドレスは0であり．プログラム・ローダは，プログラム1の起点アドレスを0から仮想記憶装置上の区域の起点に再配置する．これは，一種の再配置である．**図9-72**は，この過程を概念的に示したものである．

プログラム1は，プログラム・ローディング時にセグメント・ページ形式に変えられる．そのあとで，仮想記憶装置上のセグメント境界からロードされる．プログラムのローディング時に，プログラムのかなりの部分は，ページ枠に収容することができなくて，外部ページ記憶装置にページアウトされる．プログラム1がローディング時に再配置されたといっても，そのアドレスはいぜんとして相対アドレスである．換言するなら，仮想記憶装置のセグメント・ページ構造を反映した仮想アドレスである．このプログラムのローディングが終了すると，プログラム1の実行が開始する．動的アドレス変換機構は，プログラムの実行中に必要に応じて仮想アドレスを自動的に実アドレスに変換する．

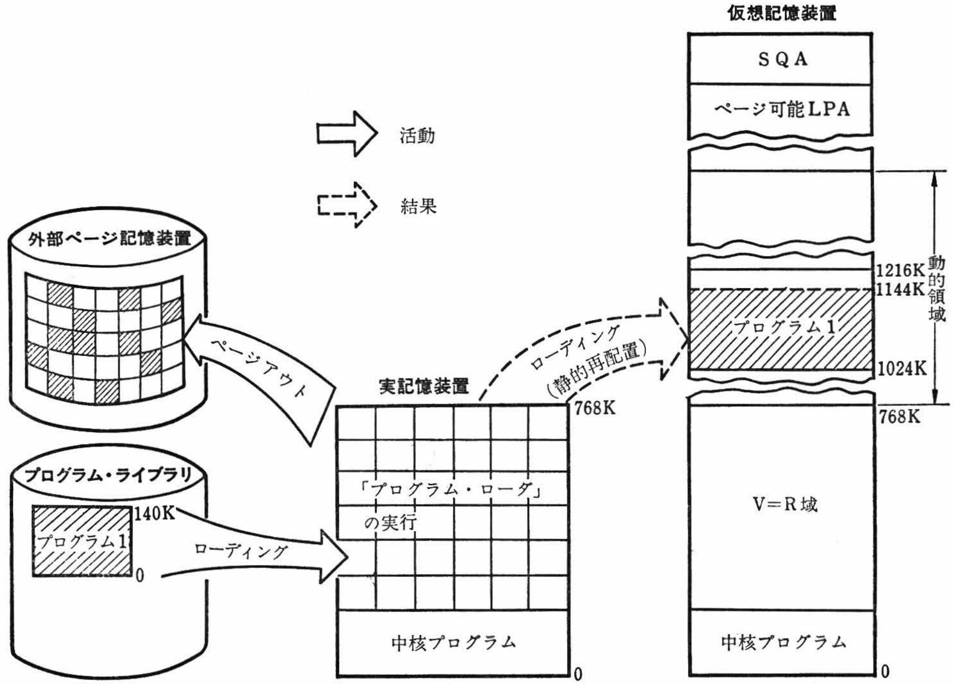


図 9-72　プログラム・ローディング

ページ不在が生じると，ページ監視プログラムは要求時ページングにもとづいて外部ページ記憶装置からそのページをロードする．

以上のように，OS/VS2は再配置と要求時ページングを使用している．プログラムは，一種の静的再配置により仮想記憶装置にローディングされる．プログラムが実行されているときに仮想アドレスが参照されると，動的アドレス変換機構を利用して，仮想アドレスを動的に実アドレスに変換する．OS/VS2は，要求時ページングの手法で，すべてのプログラム間で実記憶装置を効率的に共用させるように仕組んでいるのである．

2.　ページ監視プログラムの内部構造

仮想記憶システム用オペレーティング・システムは，従来のオペレーティング・システムを基底にし，それを拡張したものである．従来のオペレーティン

グ・システムが実記憶装置を対象にしたシステムであるのに対して，仮想記憶システムの機能を備えたオペレーティング・システムは仮想記憶装置を対象にしたシステムであるという点が著るしく異なる．すなわち，新しいオペレーティング・システムは，仮想記憶装置と実記憶装置の両方を制御しなければならないところに最大の特徴がある．ここでは，仮想記憶システムの機能を備えたオペレーティング・システムのうちの OS/VS2 を とりあげる．OS/VS2 の諸機能のうち，上述の仮想記憶装置と実記憶装置の管理をつかさどる**ページ監視プログラム**（paging supervisor）に焦点をあわせ，その機能と内部構造を検討することにする．

1） ページ監視プログラムの構造

すでに指摘したように，従来のシステムにおける主記憶装置の管理方式は，新しいシステム（OS/VS2）においてはほとんどそのまま 仮想記憶装置上の記憶域の管理方式として引きつがれている．仮想記憶装置上に存在するプログラムやデータは，物理的には外部ページ記憶装置か実記憶装置に存在することになる（**図 9–73** 参照）．これらを管理するために，ページ監視プログラムが制御プログラムの一部に追加されたわけである．

ページ監視プログラムは，基本的には要求時ページングによって仮想記憶装置と実記憶装置とを管理するものであるが，これを遂行するためにつぎのような機能を備えている：

① **アドレス可能性**（addressability）**の確立**： 仮想記憶装置上の記憶域管理に対応して，必要となるセグメント・テーブルとページ・テーブルを作成し，維持する機能．

② **要求時ページングの管理**： この機能にはつぎの４つの要素が含まれる：

- ページ不在割込みの処理
- ページ枠の使用状況の管理
- 外部ページ記憶装置上のスロットの使用状況の管理
- ページイン，ページアウトのオペレーション

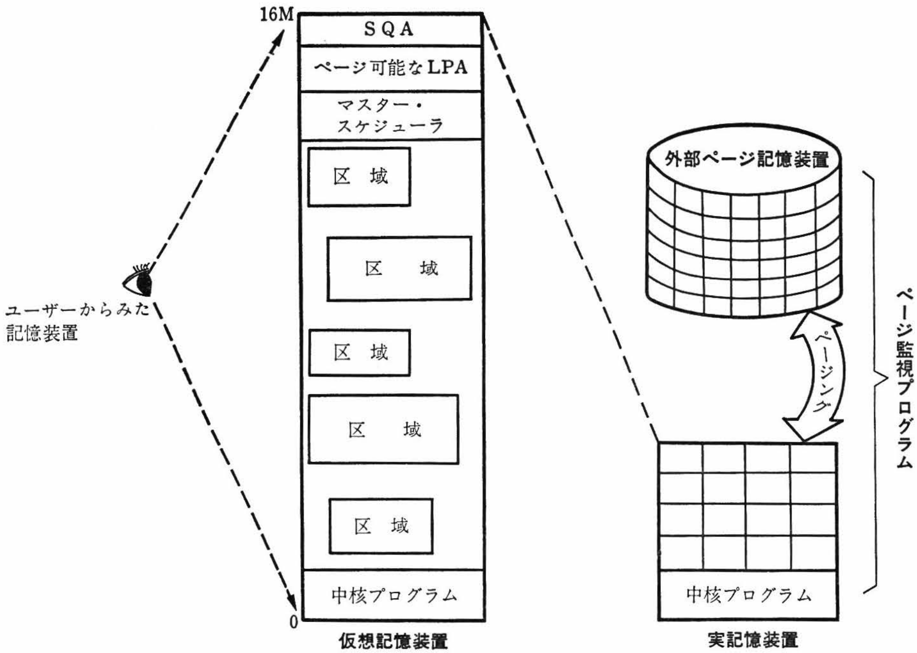


図 9-73　ページ監視プログラムの位置づけ

③　ページ固定化などのサービス機能

以上のような機能を果すために，ページ監視プログラムは**実記憶装置管理ルーチン** (real storage administrator)，**補助記憶装置管理ルーチン** (auxiliary storage administrator)，**ページ管理ルーチン** (page administrator) および**インターフェス制御ルーチン** (interface control routine) の4つの基本的なモジュールから構成されている．また，**ページ再使用** (page reclamation) の機能によって，ページインの操作を最少限にとどめるようにしている．

別の視点から，ページ監視プログラムの構造を分類すると，**非タスク・ページ監視プログラム** (non-task page supervisor). と**ページ・タスク** (page task) の2つの部分に分けられる（**図 9-74** 参照）．図において，ディスパッチャを通して実行される部分，つまり**ページ・タスク待行列走査ルーチン** (page task queue scanner) と**ページング・ルーチン** (paging routine) とがペ

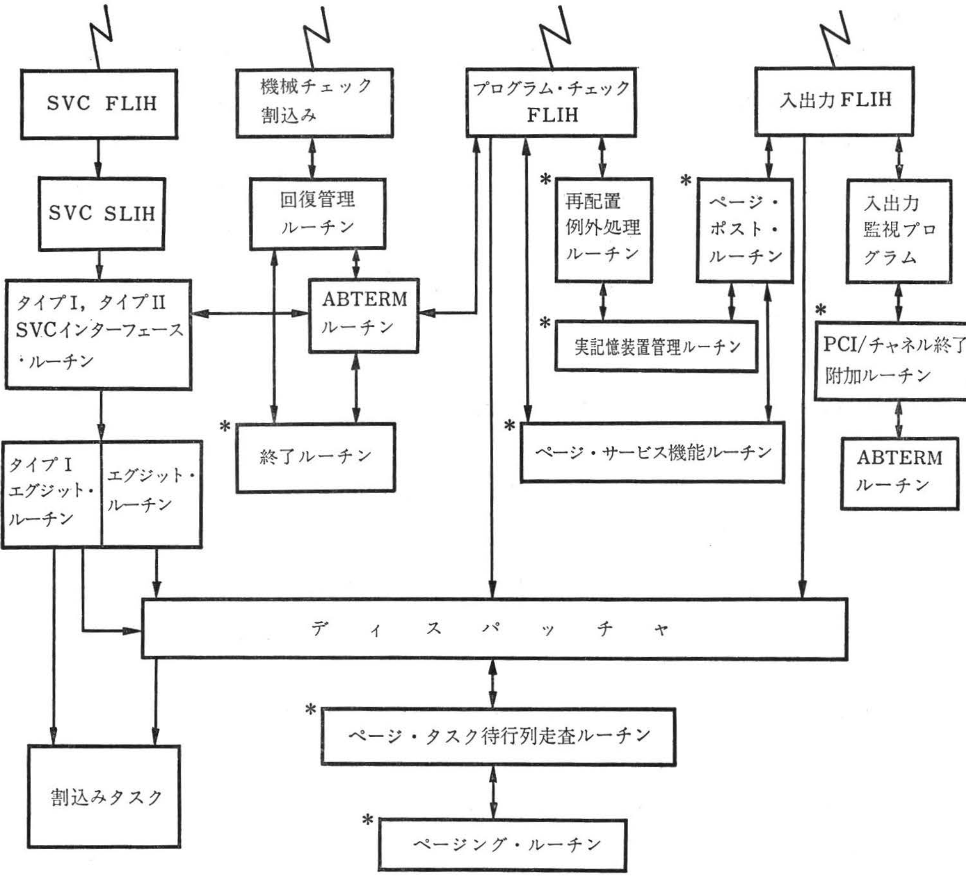


図 9-74 ページ監視プログラムの構造

ージ・タスクに属する．一方，このルーチン以外のページ監視プログラム部分が，非タスク・ページ監視プログラムに属する．

非タスク・ページ監視プログラムは，割込みが生じた場合，または何らかのサービス要求（たとえば，ページ固定の要求）が監視プログラムになされた場合に実行される．この部分は，**再配置例外処理ルーチン**（relocation exception routine)，SVC 割込み処理ルーチン，入出力割込み処理ルーチン，および**ページ・サービス機能ルーチン**（page service facility）などから構成されてい

る．

ページ・タスクは，前述したようにディスパッチャを通して実行される．ページ・タスクは外部ページ記憶装置と実記憶装置の間でのページの移動を処理する．さらに，非タスク・ページ監視プログラムがその処理の後始末をページ・タスクにまかせる場合にも機能する．ページ・タスクは，システム・タスクのうちで最も優先順位の高いマスタ・スケジューラのサブタスクとして働く．

なお，ページ監視プログラムが，制御権を受けとるのはつぎの5つの場合である（図9-74参照）：

① ページ不在割込みが生じたさいに，プログラム・チェック処理ルーチンから制御権を受けとる．
② ページアウト／ページインの必要が生じた時，ディスパッチャから制御権を受けとる．
③ ページングのための入出力操作がおこなわれた結果，**PCI**（program controlled interruption）または**チャネル終了**（channel end）の入出力割込みによって実行される．
④ ページ監視プログラムのサービスを要求するSVC命令が実行された結果，制御権を受けとる．
⑤ ページ監視プログラムのサービスを要求するブランチ命令が実行された結果制御権を受けとる．

2）ページ不在割込みの処理構造

ページ不在が生じた場合にページ監視プログラムが機能する様子をまとめたのが，図9-75，図9-76，図9-77である．**図9-75**は，ページ不在が生じたさいに非タスク・ページ監視プログラムが機能する様子である．図9-76はページングにさいしてページ・タスクが機能する様子である．また，図9-77は実際のページングのための入出力操作の面倒をみるページ入出力割込みルーチンの機能構造である．

以下で，それぞれがどのように機能するかを各図にもとづいて説明してゆ

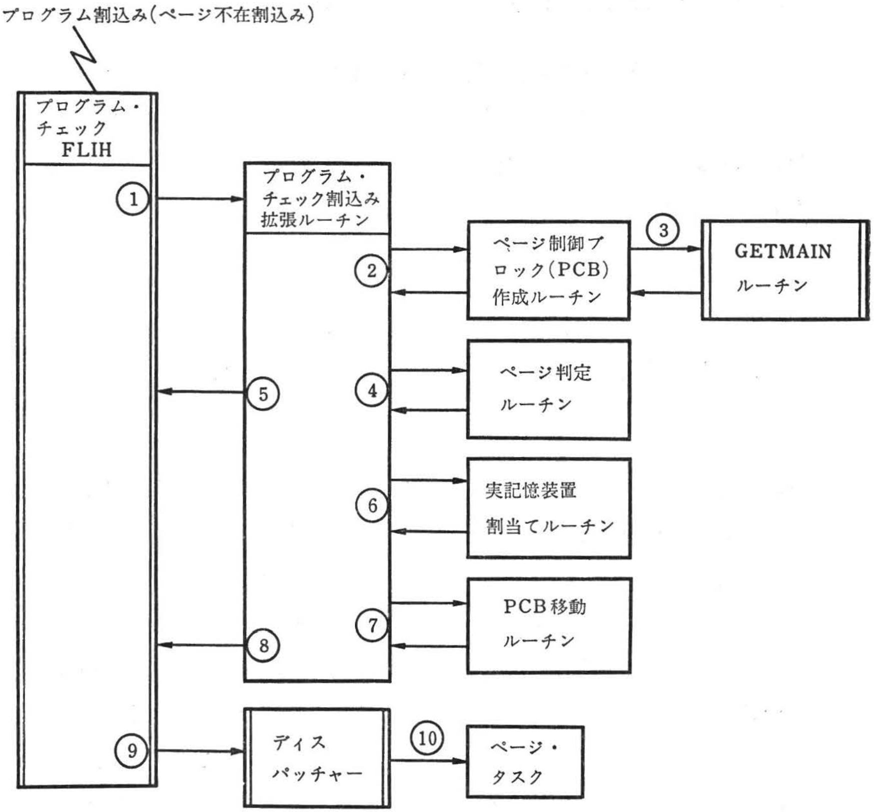


図 9-75 非タスク・ページ監視プログラムの機能構造（ページ不在割込みのとき）

く．

A．非タスク・ページ監視プログラムの機能（図 9-75 参照）

図中に示した各番号に対応して，それぞれの機能を示す．

① 実記憶装置にないページのアドレスが参照されると，ページ不在割込みが発生する．制御権は自動的にプログラム・チェック FLIH にわたる．ここで割込みコードが調べられ，ページ不在割込み（コード17）であることが判明すると，**プログラム・チェック割込み拡張ルーチン**（program check

interrupt extention routine）に制御権が移される．

② プログラム・チェック割込み拡張ルーチンは，**PCB 作成ルーチン**（build PCB module）を呼ぶ．**PCB**（Page Control Block：**ページ制御ブロック**）は，ページ不在を処理するための情報，すなわちページ番号や割込みを起したTCBのアドレスなどを貯えており，ページ監視プログラムはこのPCBの内容をチェックすることによりいかなる処理をすべきか判定する．

③ PCB 作成ルーチンは，PCB **解放待行列**（free queue）から PCB を確保する．もし，使用できる PCB が待行列に存在しないと，PCB 用の32バイトを SQA から確保する．

④ つぎに，制御権は**ページ判定ルーチン**（find page routine）に移る．このルーチンは，ページ不在割込みを起したプログラムのページ・テーブルの記入項目と外部ページ・テーブルの記入項目の中のそれぞれのアドレス（ページ枠アドレスとスロットのアドレス）をとり出してから，制御権をプログラム・チェック割込み拡張ルーチンに戻す．割込み拡張ルーチンはページ不在割込みの原因となった仮想アドレスが，そのプログラムの仮想アドレスとして妥当なものであるかどうかページ・テーブルをもとに調べる．仮想アドレスに対応するページがすでに割当てずみであれば問題ない．この場合には⑥へ行く．そうでない場合には，⑤に進む．

⑤ プログラム・チェック FLIH は，異常終止の処理をするか，SPIE エグジット・ルーチンをスケジュールする．

⑥ **実記憶装置割当てルーチン**（real storage allocation routine）へブランチする．このルーチンでは，ページ不在割込みを処理するためにページ枠を割当てる必要があるか，あるいは直ちにこの割込みを満足させることができるか（これが，ページ再使用の場合であるが），さらにはページインの操作が必要であるかどうかを調べる．その状況結果をプログラム・チェック割込み拡張ルーチンに知らせたうえで，そのルーチンに戻る．

⑦ 割込み拡張ルーチンは，**PCB 移動ルーチン**（move PCB routine）を呼び，PCB を別の待行列に移す．実記憶装置割当てルーチンがページ不在割込みの処理を完了（ページ再使用の場合）させたかどうかで PCB を入れる待行列が異なるからである．PCB の移動先としては，つぎの3つがある：

- ページ再使用がおこなわれた場合には，PCB は解放待行列に移される．
- ページインの操作が必要で，かつ実記憶装置のページ枠が割り当てられた場合には，ページ入出力開始用 PCB 待行列に移される．
- ページインの操作が必要であるが，実記憶装置上のページ枠を割り当てることができなかった場合には，割当て用 PCB 待行列に入れられる．

⑧ 割込み拡張ルーチンは，⑦の処理結果ページ不在割込み処理が完了しタスク切換えの必要がないか，あるいはさらにページ監視プログラムによる処理が必要であるかどうかを，プログラム・チェック FLIH に知らせる．

⑨ ページ不在割込み処理が完了していると，ディスパッチャを通じて割込みタスクに制御権は戻され，そのプログラムの処理が再開される．

⑩ ページ不在割込みがその場で処理できなかった場合は，そのタスクは待ち状態におかれ，ディスパッチャを通してページ・タスクに制御権が渡る．ページ・タスクではページングの処理をおこなう．また，ページ不在割込みの処理が完了している場合でも，**ページ使用可能待行列**（page available queue：後述）がある定められた値より小さくなれば，やはりページ・タスクのサービスが必要となり，この場合にもディスパッチャを経由してページ・タスクに制御権が移る．ただし，このときは割込みタスクは活動可能状態のままである．

以上が，ページ不在割込みが生じたさいの非タスク・ページ監視プログラムの機能である．

B．ページ・タスクの機能（図9-76参照）

図9-76に示したのが，ページ不在のさいに機能するページ・タスクの処理構造である．図中の番号にもとづいて，それぞれの処理機能を説明してゆこう．

① 非ページ・タスク監視プログラムで処理しきれなかった内容は，ページ・タスクによって引き続いて処理がおこなわれる．ページ・タスクはディスパッチャを通じて実行される．ページ・タスクでどのような処理をおこなう必要があるかの情報は，PCB で示されている．前述したように，PCB は処理すべき内容に応じて別々の待行列に移されている．

制御権は，まずページ・タスクのうちのページ・タスク待行列走査ルーチンに渡る．このルーチンはページ・タスクの中心的役割りを果すものであり，PCB 待行列を調べ，PCB 待行列がつぎの3つの状態のうちのどれに相当するかを判定する：

- 待行列は空である．
- 待行列は空ではないが，抑止状態にある．
- 待行列は空でなく，かつ活動可能状態にある．

待行列が抑止されていると，その待行列に関連する処理はある種の資源が利用可能になるまで待たされる．したがって，待行列が活動可能状態であると，②以降の処理に移る．

② **実記憶装置割当てルーチン**（real storage allocation routine）に制御権が移り，このルーチンで実記憶装置割当て待行列を調べる．この待行列には，ページ枠を必要とするが，割当てができなかった PCB が入れられている．実記憶装置割当てルーチンでページ枠の割り当てがおこなわれると，PCB はページ入出力開始待行列に移される．ページ枠の割り当てができないと，実記憶装置割当て待行列は抑止状態におかれ，③に進む．

③ 実記憶装置割当て待行列に PCB が存在しないか，あるいはその待行列が抑止状態であると，補助記憶装置割当て待行列が調べられる．この待行

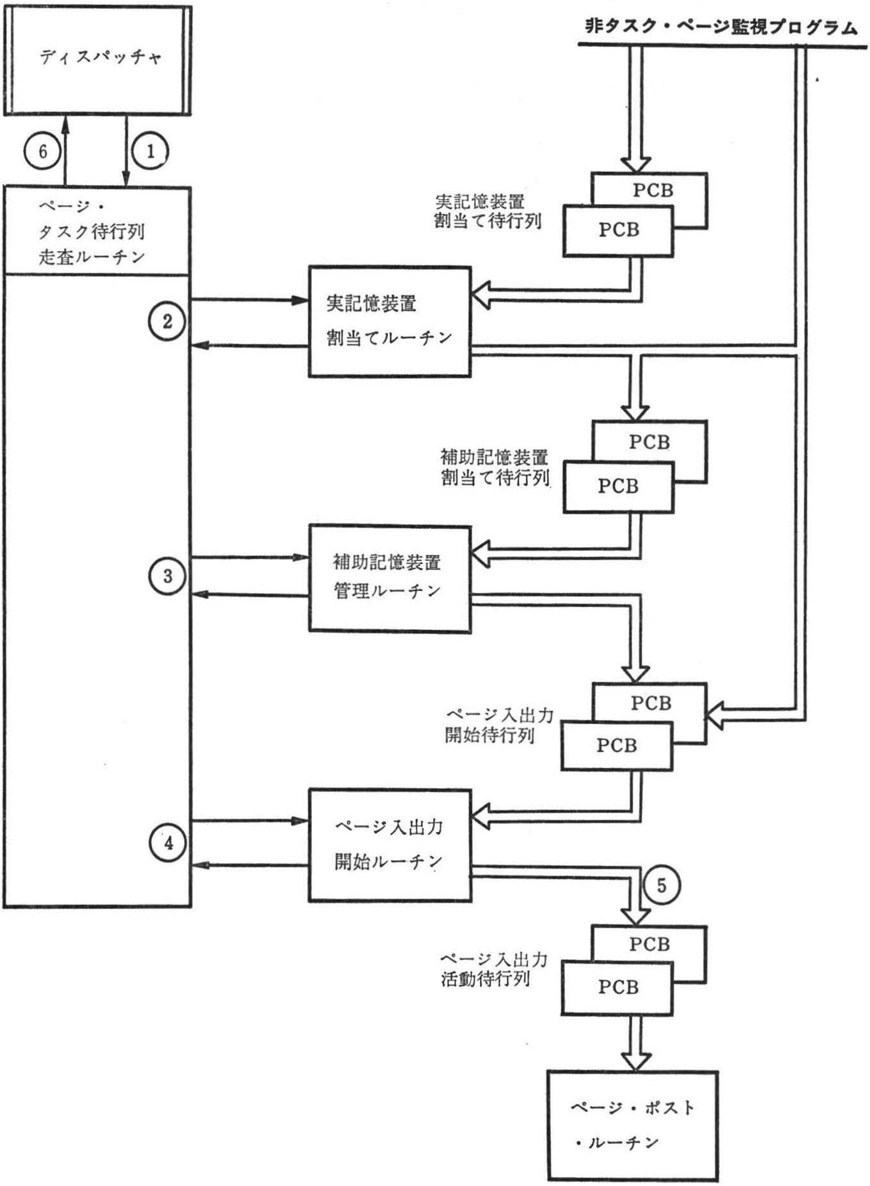


図 9-76　ページ・タスクの機能構造（ページ不在のとき）

列は，ページアウトのために，外部ページ記憶装置上の割当てに使われ，割当てが終ると待行列上の PCB はページ入出力開始待行列に移される．なお，この待行列の処理は，補助記憶装置上のページ割当てアルゴリズム（後述）にもとづいて，**補助記憶装置管理ルーチン**（auxiliary storage administrator）がおこなう．

④ 実記憶装置待行列および補助記憶装置待行列の双方が共に抑止状態にあるとか，PCB が存在しない場合にはページ入出力開始待行列が調べられる．ページ入出力開始待行列を処理するのは，**ページ入出力開始ルーチン**（page I/O initiation routine）である．なお，このルーチンは．ページ入出力スロット分類アルゴリズム（後述）を用いて，ページ入出力の効率の向上をはかっている．

⑤ ページ入出力スロット分類アルゴリズムの実行後，ページ入出力開始ルーチンは PCB 移動ルーチンを呼ぶ．PCB 移動ルーチンは，ページ入出力開始待行列から，ページ入出力活動待行列へ PCB を移す．

ページ入出力活動待行列は，ページ・ポスト・ルーチンで処理される．使用可能なチャネル・プログラムがない場合には，ページ入出力開始ルーチンがページ入出力開始待行列を抑止状態におく．

⑥ すべての待行列が処理されたあと，ページ・タスクは待状態におかれ，制御権はディスパッチャに戻される．

以上が，非ページ・タスク監視プログラムからディスパッチャを経由して制御権が移されたさいのページ・タスクの処理機能である．

C．ページ入出力割込みの処理機能（図 9-77 参照）

ページングのための入出力が完了して割込みが生じると，図 9-77 で示しているような処理手順がとられる．図中の番号にしたがって，それぞれの処理内容を以下で説明する：

① 入出力割込みが生じると，入出力 FLIH に制御権が渡る．

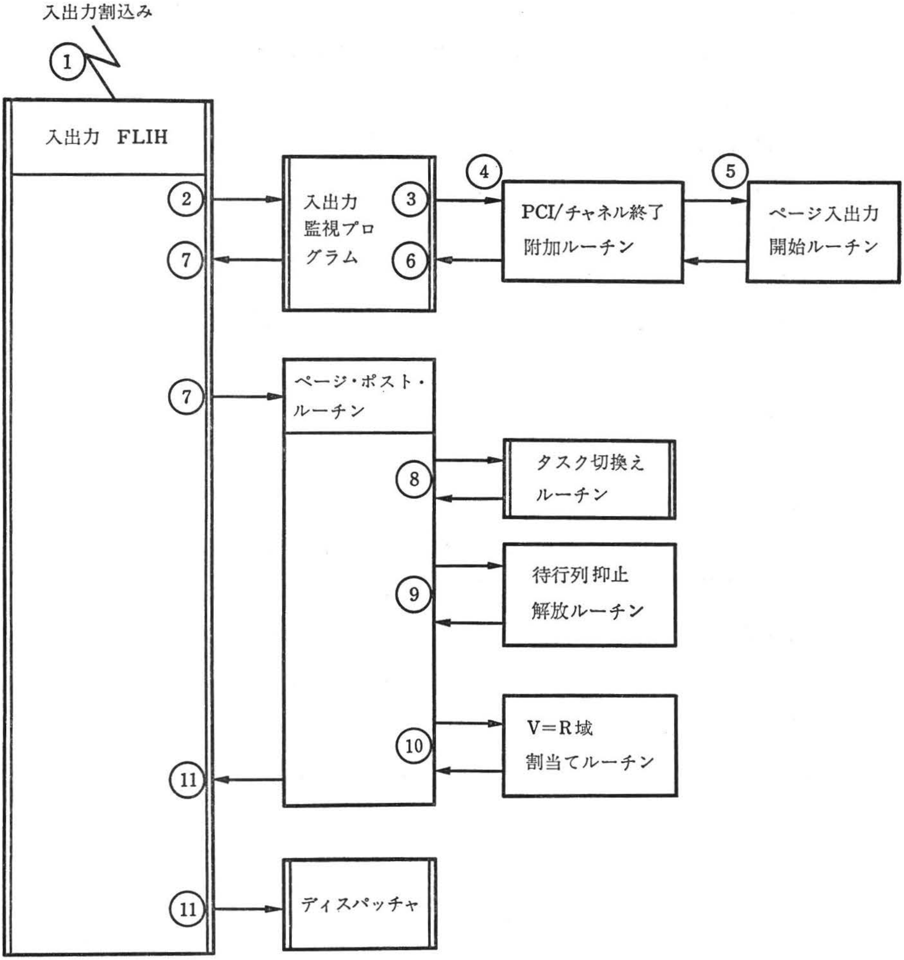


図 9-77 ページ入出力割込みの処理構造

② 割込み処理のために，さらに入出力監視プログラムに制御権が移る．入出力割込みが，ページ監視プログラムが作りあげたチャネル・プログラムの実行によって生じたのであれば，制御権は **PCI/チャネル終了附加ルーチン** (PCI / channel end appendage routine) に移る．

③ と ④ PCI / チャネル終了附加ルーチンに制御権が移ると，このルーチンは完了したチャネル・プログラムに関連する PCB に完了標識を立て，

ポスト・ビットをセットする．この時点で，待ち状態のチャネル・プログラムがあると，その実行をスケジュールするために，ページ入出力開始ルーチンを呼ぶ．

⑤ ページ入出力開始ルーチンは，後述するページ入出力スロット分類アルゴリズムのステップ3と4を実行したうえで，PCI/チャネル終了附加ルーチンに戻る．

⑥ ページ入出力開始ルーチンが外部ページ記憶装置を相手にページングを開始したら，入出力監視プログラムはその処理をおこない，入出力 FLIH に戻る．

⑦ 入出力 FLIH が入出力監視プログラムから制御権を受けとると，ポスト・ビットを調べる．このビットがオンだと，ページ・ポスト・ルーチンに制御権を移す．

⑧ ページ・ポスト・ルーチンは，ページ入出力活動待行列を調べ，PCB の中に完了標識が立っているかどうかチェックする．チェックされた PCB がページ不在の割込みの結果作られたものであると，完了標識が立っていることは，該当ページが実記憶装置のページ枠にページインされたことを意味する．したがって，ページ・テーブルの該当記入項目の内容を更新する．そのあとで，タスク切換えルーチンが機能し，TCB の新ポインタを変更する．

⑨ 資源の供給，つまり実記憶装置のページ枠を確保するためにページアウトをおこなった結果 PCB に完了標識が立てられていると，**待行列抑止解放ルーチン** (release queue supression routine) が呼び出される．このルーチンは，実記憶装置のページ枠の割り当てができなかったために抑止状態になっている PCB 待行列を活動可能状態にもどすと同時に，ページ・タスクを作動可能にする．

⑩ V=R 域内にある実記憶装置を解放するための完了であると，V=R 域内にページを割り振るために **V=R 域割当てルーチン** (V=R region allocation routine) が呼ばれる．

⑪ ページ・ポスト・ルーチンは，全ての PCB を処理し終ると，ポスト・ビ

ットをリセットし，入出力 FLIH に制御権を戻す．入出力 FLIH はディスパッチャにブランチする．

これまでみてきたように，非タスク・ページ監視プログラム，ページ・タスク，およびページ入出力割込みの処理機能の相互連係によって，ページ不在の割込みが生じた場合の処理が完遂されるのである．

3) 実記憶装置管理ルーチンの機能構造

実記憶装置管理ルーチン（real storage administrator）は，実記憶装置上にあるすべてのページ枠の管理を担当し，実記憶装置の割当て，解放，および状況の変更をおこなう．これらの機能を大別するとつぎの5つに集約できる：

A．実記憶装置の割当て機能
B．SQA/LSQA の割当て機能
C．V=R 域の割当て機能
D．ページ置換機能
E．タスクの実行中断の機能

以下で，機能ごとにそれぞれの処理内容と構造を検討してゆこう．

A. 実記憶装置の割当て機能

ページ不在の割込みを処理したり，ページの長期固定化の要求を満たしたり，PCB 上で延期されている要求を満足させるために，実記憶装置の割当て機能がはたらく．実記憶装置上のページ枠を確保する場合，まず**ページ再使用**（page reclamation）を試みる．つまり，ページ枠自体はすでに解放されているが，その場所にまだ別のページが読み込まれていないページ枠をページ枠テーブルの中で捜し，必要とするページがあるかどうかを確認する．もし，要求ページがそのままの状態で残っていると，わざわざページング入出力操作をおこなわなくても，ページ不在を満すことができる．必要とするページが見つからないとか，最初のページ参照である場合には，ページ枠テーブルの中の使用

可能ページ待行列（後述）の中からページ枠の割当てをおこなう．この場合はページインの操作が必要となる．したがって，該当 PCB をページ入出力開始待行列に移す．

後述するように，現在使用中のページ枠を使用可能ページ待行列に供給する必要があるときは，ページ置換アルゴリズムが適用される．また，ページの固定化要求があると，そのページ枠にはできるだけ V=R 域と重複しないページ枠を割り当てるようにしている．

B．SQA/LSQA の割当て機能

実記憶装置上のページ枠を SQA あるいは LSQA として使いたい場合には，**SQA/LSQA 割当てルーチン**（SQA/LSQA allocation routine）が呼び出される．

これらの区域に対してページの長期固定化の要求がなされていると，SQA/LSQA ルーチンは V=R 域以外の場所に必要なページ枠を確保するよう試みる．というのは，V=R 域の要求があったときに，競合状態が発生してしまうのを極力回避したいからである．この場合の選択基準はつぎの通りである：

① ページ枠が V=R 域以外の場所にとれれば，その場所を割り当てる．

② 使用可能待行列の中のページ枠のうち，V=R 域以外でかつ変更ビットがオフのページ枠からとる．

③ 保留待行列の中のページ枠のうち，V=R 域以外でかつ変更ビットがオフのページ枠からとる．

以上のいずれも駄目であると，

④ 上記①から③の順で V=R 域の中のページ枠からとる．

⑤ まだページインが実行されていない PCB を捜し，その PCB に対応するページ枠を割り当てる．その結果，該当 PCB は実記憶装置割当て待行列にもどされる．

C．V=R 域の割当て機能

この場合は，仮想記憶装置上で割当てられている記憶域のアドレスと同じ実記憶装置上の区域を割り当て，しかもページの長期固定化をおこなう．したがって，仮想記憶装置上の区域と実記憶装置上の区域とは全く1対1で対応する．これによって，V=R 域を使用したジョブの場合にはチャネル・プログラムの変換は不要となる．

D．ページ置換機能

ページ不在を起したページでページ再使用が不可能なものに対しては，ページ枠を割り当てる必要がある．これをおこなうためには，目下未使用のページ枠がどこにあるかを表示したリストを保持しておくことが必要だ．このリストにあたるのが，**使用可能ページ待行列**（available page queue）である．**使用可能ページ枠数**（Available Page frame Count ；略して **APC**）が，ある定められた値（最低限界値）よりも低くなると，ページ監視プログラムは，目下使用中のページ枠の中から適当なものを選んで，使用可能ページ待行列に供給する．使用可能ページ待行列に移されたページ枠は，それ以降どのプログラムに割り当てられてもよい状態になってしまうので，そのページ枠の内容が変更されている場合には，該当ページを外部ページ記憶装置にページアウトしたうえで，使用可能ページ待行列に移される．ページ枠の内容が変更されていない場合だと直接使用可能ページ待行列に移される．なお，この待行列へのページ枠の供給は，APC がある定められた値（最高限界値）に達するまでつづけられる．

ページ監視プログラムは，目下使用中のページ枠からどのページ枠を選択して使用可能ページ待行列へ移すかをきめるさいに，それ以降のページング操作が最少限にとどめられるような配慮をしている．これが，**ページ置換アルゴリズム**（page replacement algorithm）である．このアルゴリズムは，当然のことながら LRU 方式にしたがっている．つまり，使用可能ページとしては参照頻度の少ないページを選ぶようにし，しかもページアウトを最低限におさえるために変更ページは極力選ばないといった基準をもとにした LRU 方式をとっている．このようなページ枠の選択にあたっては，参照ビットと変更ビットを利

用する．すでに指摘したように，これらのビットは記憶保護キーの一部であり，CPU がそのページ枠を参照したり，変更したりすると自動的にハードウェアでセットされる．実記憶装置管理ルーチンは，これらビットを調べることによって，参照頻度の少ないページを選ぶ．さらに，ページ枠を選ぶ場合，できるだけ変更されていないページ枠を選ぶことによって，ページアウトの回数を少なくするようにしている．以下でページ置換アルゴリズムを具体化してゆこう．

OS/VS 2 のページ置換アルゴリズムでは，ページ枠テーブルを6種類の待行列に区分してコントロールしている．6つの待行列とはつぎのものである：

① **使用可能ページ待行列**（あるいは，単に **使用可能待行列**）：すぐに割り当てることのできるページ枠の集まり．

② **保留待行列**（hold queue）：割当てがなされた直後のページ枠の集まりであり，この待行列にあるページ枠は割当ての対象にしない．すなわち，割り当てられたばかりのページは，近い将来必ず参照されると想定して，ページングの対象としない方法をとっている．

③ **活動待行列**（active queue）：この待行列にあるページ枠はすでに割当てずみで，プログラムで使用されているが，再割当ての対象となる可能性があるものの集まりである．参照ビットと変更ビットのセット状況の組合せで，4種類の待行列に分かれている．4つの待行列とは，つぎのものである．この場合，前の0または1は参照ビットを示し，後の0または1は変更ビットを示している．

- （0，0）：該当ページは参照も，変更もされていない．
- （0，1）：ページ監視プログラムが前回調べた時点より以降，該当ページ枠は参照されてはいないが，それ以前の時点で変更されている．
- （1，0）：該当ページ枠は参照されたが，その内容は変更されていない．
- （1，1）：該当ページ枠は参照もされ，その内容に変更も加えられ

た．

ページ監視プログラムは，参照ずみのページ枠を選ぶ前に，参照されたことのないページを選び，またその内容が変更されたページを選ぶ前に，変更が加えなれていないページを選択する．各々のページ枠は，参照ビットと変更ビットの状況により，上記の4つの待行列のどれかに割り振られることになる．そして，それぞれの待行列は，先入れ-先出し法（FIFO）を基準として処理される仕組みになっている．

幾つかのページ枠が要求され，使用可能ページ待行列の中に十分なページ枠がない場合，ページ監視プログラムは，まず（0，0）待行列に登録されているページ枠の中からページ枠を選ぼうとする．そのさい，記憶保護キーを読みとり，参照ビットが“0”のものを使用可能ページ待行列に移す（もし，変更ビットが“1”だと，ページアウトにしたうえで使用可能ページ待行列に入れる）．参照ビットが“1”になっているページ枠に関しては，（1，0）または（1，1）の待行列に移すが，このとき参照ビットは“0”にする．この結果，最も最近に参照されたページ枠は，活動待行列に依然として保管することができる．このとき，変更ビットはリセットされない．

以上の一連の操作は，最高限界までページ枠が使用可能ページ待行列に移されるまでくり返される．この様子を具体例にもとづいて，図9-78から図9-80までを参照しながら詳しく順を追って説明してゆこう．これら図中，PFTEは，**ページ枠テーブルの記入項目**（page frame table entry）の略であり，Rは**参照ビット**（reference bit），Cは**変更ビット**（change bit）を意味する．以下で図中の番号順にページ置換のアルゴリズムを具体的にしよう：

① 全てのページ枠が使用可能待行列にある（最初の状態）．

② ページ監視プログラムがページ枠を割り当てると，そのページ枠は保留待行列に移され，プログラムの実行が開始される．このとき，活動待行列はまだ空である．**図9-78**の②では，ページ枠の1から10までが割り当てられたためAPCは2となっている．使用可能待行列の**最低限界値**（low threshold）を3とし，**最高限界値**（high threshold）を5とすると，この場合APCは最

低限界値より小さくなっているので，使用可能待行列への値が最高限界値に等しくなるようページ枠を供給する必要がある．

③　保留待行列全体が，活動待行列（0，0）に移される．

④　活動待行列（0，0）の上部から順番に，未参照で，かつまだ変更されていないページ枠を捜し，使用可能待行列に移す．**図 9-79** では，ページ枠 2，5，7を使用可能待行列に移動することで最高限界値が満されるので，ページ枠の探索はこれでいったん終る．探索したページ枠のうち（0，0）状態のページ枠以外のものは，参照ビットを0にしたうえでそれぞれ該当する活動待行列に移される．つまり，ページ枠1および4は活動待行列（1，0）へ，またページ枠3と6は活動待行列（1，1）に移される．

⑤　今度は，ページ枠11，12，2が割り当てられたとすると，これらのページ

①

使用可能待行列 R	C		保留待行列 R	C		活動待行列 (0,0) R	C		活動待行列 (0,1) R	C		活動待行列 (1,0) R	C		活動待行列 (1,1) R	C	
0	0	PFTE 1															
0	0	2															
0	0	3															
0	0	4															
0	0	5															
0	0	6															
0	0	7															
0	0	8															
0	0	9															
0	0	10															
0	0	11															
0	0	12															

②

0	0	PFTE 11	0	0	PFTE 1												
0	0	12	0	0	2												
			0	0	3												
			0	0	4												
			0	0	5												
			0	0	6												
			0	0	7												
			0	0	8												
			0	0	9												
			0	0	10												

③

0	0	PFTE 11				1	0	PFTE 1									
0	0	12				0	0	2									
						1	1	3									
						1	0	4									
						0	0	5									
						1	1	6									
						0	0	7									
						0	0	8									
						1	1	9									
						1	0	10									

図 9-78　ページ置換アルゴリズム (1)

④

使用可能待行列	保留待行列	活動待行列 (0,0)	活動待行列 (0,1)	活動待行列 (1,0)	活動待行列 (1,1)
R C	R C	R C	R C	R C	R C
0 0 PFTE11		0 0 PFTE 8		0 0 PFTE 1	0 1 PFTE 3
0 0 12		1 1 9		0 0 4	0 1 6
0 0 2		1 0 10			
0 0 5					
0 0 7					

⑤

使用可能待行列	保留待行列	活動待行列 (0,0)	活動待行列 (0,1)	活動待行列 (1,0)	活動待行列 (1,1)
0 0 PFTE 5	0 0 PFTE11	0 0 PFTE 8		1 0 PFTE 1	1 1 PFTE 3
0 0 7	0 0 12	1 1 9		1 1 4	0 1 6
	0 0 2	1 0 10			

⑥

使用可能待行列	保留待行列	活動待行列 (0,0)	活動待行列 (0,1)	活動待行列 (1,0)	活動待行列 (1,1)
0 0 PFTE 5	0 0 PFTE11			1 0 PFTE 1	1 1 PFTE 3
0 0 7	0 0 12			1 1 4	0 1 6
0 0 8	0 0 2			0 0 10	0 1 9

図 9-79　ページ置換アルゴリズム　(2)

枠は必然的に保留待行列に移されて，この結果，APC は再び最低限界値未満になるので，使用可能待行列へのページ枠の供給が必要となる．

ページ枠11，12，9は保留待行列に移されたばかりなので，これらのページ枠は供給の対象にされない．これは，参照されたばかりのページ枠は，近い将来また参照されるであろうという仮定にもとづいている．

活動待行列（1，0）上のページ枠1，4は参照ビットおよび変更ビットがさきの状態から変化している．また，活動待行列（1，1）上のページ枠3の参照ビットも変っている．これらは，ステップ④のあとでプログラムの実行によって参照されたとみなされる．

⑥ ページ監視プログラムは，3つのページ枠を使用可能待行列に提供する必要に迫られている．まず，活動待行列（0，0）を探索する（図 9-79 の⑤

参照）．その結果，ページ枠８は使用可能待行列へ，ページ枠10は活動待行列（1，0）へ，ページ枠９は活動待行列（1，1）へ参照ビットを０にしたうえで移される．この状態を示したのが図 9-79 の⑥である．なお，ステップ⑥では，まだ最高限界値より低いので，さらに監視プログラムはページ枠の供給を続ける．その様子を示したのが，**図 9-80** の⑦，⑧，⑨である．

⑦　ページ監視プログラムは，引き続いて活動待行列（0，1）を調べるが，この待行列は空である（図 9-79 の⑥参照）．ページ監視プログラムは，使用可能待行列へのページ枠の供給は必ず活動待行列（0，0）および（0，1）から実施する．したがって，活動待行列（1，0）全体を活動待行列（0，0）へ，活動待行列（1，1）全体を活動待行列（0，1）へ移動させる．さらに，保留待行列全体を活動待行列（1，0）に移す．この状況を示した

⑦

使用可能待行列 (R C)	保留待行列 (R C)	活動待行列 (0,0) (R C)	活動待行列 (0,1) (R C)	活動待行列 (1,0) (R C)	活動待行列 (1,1) (R C)
0 0 PFTE 5		1 0 PFTE 1	1 1 PFTE 3	0 0 PFTE 11	
0 0 7		1 1 4	0 1 6	1 1 12	
0 0 8		0 0 10	0 1 9	1 0 2	

⑧

使用可能待行列	保留待行列	活動待行列 (0,0)	活動待行列 (0,1)	活動待行列 (1,0)	活動待行列 (1,1)
0 0 PFTE 5			1 1 PFTE 3	0 0 PFTE 11	0 1 PFTE 4
0 0 7			0 1 6	1 1 12	
0 0 8			0 1 9	1 0 2	
0 0 10				0 0 1	

⑨

使用可能待行列	保留待行列	活動待行列 (0,0)	活動待行列 (0,1)	活動待行列 (1,0)	活動待行列 (1,1)
0 0 PFTE 5			0 1 PFTE 9	0 0 PFTE 11	0 1 PFTE 4
0 0 7				1 1 12	0 1 3
0 0 8				1 0 2	
0 0 10				0 0 1	
0 0 6					

図 9-80　ページ置換アルゴリズム　(3)

のが，⑦である.

⑧ このあとで，活動待行列（0，0）から調査を再開する．この結果，ページ枠1は活動待行列（1，0）へ，ページ枠4は活動待行列（1，1）へ，さらに，ページ枠10は使用可能待行列に移動する．この状態では，APC=4となり，まだ使用可能状態のページ枠が不足している．この状況を示したのが，図9-80の⑧である.

⑨ ステップ⑧における活動待行列（0，1）を調べる．この待行列にあるページ枠の変更ビットは“1”にセットされている．つまり，この待行列から使用可能待行列へページ枠を供給する前に，外部ページ・データ・セット上のスロットに該当ページをページアウトする必要がある．ページ枠3は活動待行列（1，1）へ，ページ枠6はページアウトの実行後，使用可能ページ待行列に移す．この時点でAPC=5となり，ページ監視プログラムはその働きを止める．この状態がステップ⑨である.

以上みてきたようなアルゴリズムでページ置換を遂行しているのがOS/VS2である．すでにこれまでの説明で明らかなように，OS/VS2でもLRU法にもとづいたページ置換を行なっている.

E. タスク実行不能アルゴリズム

これまで説明してきたように，使用可能待行列に収容するページ枠は原則として，活動待行列(0，0)と(0，1)から捜しだす．しかし，ある段階ではこれらの待行列の中に十分なページ枠が見つからないで，活動待行列(1，0)や（1，1）を捜しにゆく頻度が多くなる状態が考えられる．待行列(1，0)と（0，1）には比較的近い時点で参照されたページ枠が収容されているので，この状態を放置しておくと，ページング・オペレーションが繁発し，システムの効率をさげてしまう可能性がある.

したがって，何らかの方法でページング・オペレーションを一定の率に押さえる必要がある．このために，OS/VS2では一定期間中のページ枠の要求回数とページング入出力によるシステム待ち時間をもとにして，システムのページ

ング率が過多であるかどうかの判断をおこなっている．その判断結果，ページング率が一定基準を越えていると，指名順位の低いタスクを**実行不能状態**（disable）にすることによって，ページング率をさげる方法をとっている．この方法を，一般に**タスク実行不能アルゴリズム**（task disable algorithm）と呼んでいる．

このアルゴリズムの働きによって，つぎの利点が得られ，仮想記憶システムの柔軟な運用が可能となる：

- オンラインのトランザクションの多い時間帯では，バッチ処理用のタスクが実行不能状態におかれ，逆にトランザクション量が少くなるにしたがって，システムの能力に見合って，バッチ処理タスクが実行可能状態にもどされる．この結果，動的なローディング調整機能が発揮される．
- オペレータが，システムの能力をおしはかりながらジョブを開始させる必要はなく，システムが自動的に効率的に働くよう制御してくれる．

以下で，一定期間中のページ枠の要求回数とページング入出力によるシステム待ち時間のそれぞれにもとづいたタスク実行不能化の方法についてふれておこう．

① ページ枠の要求率にもとづいたタスク実行不能化

使用待行列からページ枠が要求されるつど，要求回数に1を加える．一定時間が経過すると，この要求回数とシステムが持っている2つのパラメータ，つまり**最低要求回数**（low reclaim count）と**最高要求回数**（high reclaim count）とを比較する．

比較結果，要求回数が最高要求回数を越えている場合には，**休業回数**（shut-down count）と呼ばれるフィールドに1を最加算したうえで，最も指名順位の低いタスクをディスパッチ不能状態におく．

一方，要求回数が最低要求回数よりも小さく，かつ休業回数が正であるときには，休業回数から1を引いたうえで，ディスパッチ不態状態のタスクの中から最も指名順位の高いタスクをディスパッチ可能状態に戻す．

なお，要求回数の比較が終るつど，その回数はリセットされる仕組みとなっている．

② ページング入出力によるシステム待ち時間にもとづいたタスク実行不能化

この場合には，**システム・ページ読み込み回数**（system page read count），**タスク・ページ読み込み回数**（task page read count），**ページ待ちタスク回数**（page wait task count）および**総ページ待ち時間**（tatal page wait time）をもとに，システムの平均待時間を算出し，この平均待時間が定められた基準時間を越えているかどうかで，タスクを実行不能状態にするかどうか判定している．

以下でその手順を説明しておこう：

- **システム・ページ読み込み回数のセット：** ページイン・オペレーションを行なうために，ページ入出力待行列に PCB が移されるつど，実記憶装置割当てルーチンが 1 を加算する．
- **タスク・ページ読み込み回数のセット：** 実記憶装置割当てルーチンは，システム・ページ読み込み回数が増加するごとに，PCB から対応する TCBをみつけ出し，タスクごとに用意されたタスク・ページ読み込み回数に 1 を加算する．
- **ページ待ちタスク回数のセット：** プログラム・チェック FLIH は，ページ不在のためにタスクが待ち状態になるつど，ページ待ちタスク回数に 1 を加える．一方，ページ・ポスト・ルーチンは，ページインが完了し，タスクが作動可能状態にもどるつど，ページ待ちタスク回数から 1 を引く．
- **システム待ちタスク**（system wait task）**にディスパッチした時：** 作動可能状態のタスクがなくなり，システム待ち状態に入るまえに，システム待ちタスクがディスパッチされる．このタスクでは，ページ待ちタスクの回数をチェックし，正の時にはその時点のタイマー値（TOD）を記録し，ページ待ち時間を測定中であることを示すスイッチをオンにする．
- ディスパッチャに入るつど，ページ待ち時間測定の表示スイッチがオン

かオフかを調べる．オンだと，その時点のタイマー値と，システム待ちタスクが記録したタイマー値との差を総ページ待ち時間に加算する．

- n回目のページ監視プログラムの**統計間隔**（statistics interval）ごとに，システム・ページ読み込み回数を調べる．

この回数が，$n\times5$ であるとページングあたりの平均待ち時間をつぎの計算式で算出する：

$$\text{平均待ち時間} = \frac{\text{（総ページ待ち時間）}}{\text{（システム・ページ読み込み回数）}}$$

この平均待ち時間が，システムで定められている最大ページ待ち時間よりも大きい時に，タスク・ページ読み込み回数が最も大きいタスクをディスパッチ不能状態にする．

以上のようなアルゴリズムで，ページング・オペレーションの繁発を防止し，システム効率がさがらないようにしているのである．

4)　補助記憶装置管理ルーチンの機能構造

補助記憶装置管理ルーチン（auxiliary storage administrator）は，外部ページ記憶装置として使用される補助記憶装置（固定ヘッドおよび非固定ヘッドの両方）の管理を行なう．また，ページ・データ・セットのために割り当てられた補助記憶装置上のレコードを管理し，ページアウト・オペレーションを効率的に行うようなページ・スロットの割り振りをする．補助記憶装置上にページ・スロットを割り振るさいのアルゴリズムは，補助記憶装置の使用度のバランスをとり，回転待時間を最少にすることを配慮したものである．

以下で，**補助ページ・セット**（auxiliary page set）の形式とページ割当てアルゴリズムについて説明することにする．

A. 補助ページ・セットの形式

補助ページ・セットとは，ページ・データ・セットに適用される補助記憶装置上の記憶域であり，１つ以上の順次データ・セットから構成される．また，

どんな補助記憶装置の組み合わせで補助ページ・セットが構成されていてもかまわない．

補助ページ・セットのための補助記憶装置の容量は，使用される仮想記憶装置の容量を基準に定められる．つまり，中核プログラムと SQA 用の記憶域を除いた部分の仮想記憶装置の各ページが収容できるだけの補助記憶装置上の容量が必要となる．

補助ページ・セット用データ・セットのいくつかを**主ページ装置**（primary page device）とし，残りのデータ・セットを**副ページ装置**（secondary page device）に区分することによって，ページング・オペレーションの最適化をはかってくれる．つまり，利用可能なスペースがある決められた値以下にさがるまで，主ページ装置がページング・オペレーションに適用される．主ページ装置上の利用可能スペースが最低限界値以下になると，最も低い指名順位のタスクで占められている主ページ装置上の各ページを副ページ装置上に移す．このオペレーションのことを，**マイグレーション**（migration）と呼んでいる．このためには，主ページ装置には磁気ドラム記憶装置のような高速装置を，副ページ装置には通常の磁気ディスク記憶装置を適用した場合に効果的であることはいうまでもない．なお，主ページ装置として使いたい装置は，システム生成時または IPL 時に指定することができる．もし，この種の指示がなされていないと，中核プログラムの方で自動的に主ページ装置を割り当ててくれる．この場合には，固定ヘッドの装置が主ページ装置として選択されるようになっている．

ここで，補助ページ・セットの形式についてふれておこう．

補助ページ・セット用のデータ・セットは，グループ番号とページ番号とでページ・レコードがアクセスできるような形式になっている．グループ番号では，グループを構成している1ないしは複数個のトラックを表示する．グループを構成するトラック数は，使用する装置によって異っている（**表 9-1** 参照）．それぞれのグループは，定められた数のスロットを収容する．スロットは，ページの大きさに等しい物理レコードであり，各グループ内で1からNまでの番号が与えられている（Nは，グループ内のスロット数に等しい）．補助ページ

表 9-1　グループ，スロットの特性（装置例ごと）

補助記憶装置	A	B	C	D
グループ当りトラック数	1	4	1	5
グループ当りスロット数	3	13	3	8
シリンダ当りグループ数	8	2	19	4
シリンダ当りバイト数	93,304	106,496	233,472	131,072
装置当りの容量(バイト数)	4,694,016	10,170,368	94,076,928	26,181,632
最大ページ数	1,146	2,483	22,968	6,392
装置当り最大グループ数	382	191	7,656	799

・セットの形式に関して，装置ごとにまとめたのが**表 9-1** である．

B．補助記憶装置上のページ割当てアルゴリズム

ページアウトのオペレーションを行う場合，どの補助記憶装置を対象にページアウトするか，またその装置上のどの場所を使用するかを決めるのに，補助記憶装置上の**ページ割当てアルゴリズム**（auxiliary storage page assignment algorithm）を適用する．さらに，このアルゴリズムは，ページ・スロットの決定には，**スロット分類アルゴリズム**（slot sorting algorithm：後述）を使用する．

外部ページ・セット用データ・セットは，前述したようにグループとスロットに形式化される．したがって，補助記憶装置上のページ・レコードは，装置番号，グループ番号およびスロット番号を使ってアクセスできる．以下で，ページング装置の決定，スロット番号の決定およびグループ番号の決定がどのような基準でおこなわれるかみておこう．

① ページング装置の決定

すでにふれたように，補助記憶装置上のページ・スロットの選択は，主ページ装置からおこなわれる．同じタイプの装置が複数個あるときは，ページング・オペレーションのバランスをとるために，利用可能なページ・スロットが沢山残されている方を選択する．

主ページ装置に割り当てることができなくなると，マイグレーションがおこ

る．どの区域のプログラムをマイグレーションするかは，ディスパッチャが決定する．区域が決まると，その区域を構成する各ページが，主ページ装置から副ページ装置へ移される．いったん，副ページ装置にマイグレーションされると，その区域のページはそのタスクの実行が終るまで副ページ装置に留まったままである．

② スロット号等の決定

最後に割り当てが行われたスロット番号のつぎのスロット番号（これを**主スロット**というが）をとる．補助ページ・セット上のあるスロットと，つぎのスロット（主スロット）とは，一回転内で連続してアクセスできるからである．

③ グループ番号の決定

グループ番号の決定にあたっては，適用装置が固定ヘッドか移動ヘッドかによって処理が異なる：

- **固定ヘッドの場合**； 装置番号と②で決定したスロット番号を使って，外部ページ・テーブル上の該当するスロットの記入項目を調べ，最初の利用可能なページ（グループ）を割り当てる．該当するスロットに利用可能なページがないと代替スロットから捜す．
- **移動ヘッドの場合**； 最後に使われたスロットが属するグループ番号が使われる．そのグループが属しているシリンダ内のグループについて，該当のスロット（②で決定したスロット）が利用可能かどうか調べる．利用可能であれば，そのページ・スロットを使う．利用できないとすると，そのスロットの代替スロットを捜しに行く．代替スロットにも利用可能なページがない場合に，はじめて外部ページ・テーブルを用いて，利用可能なページを捜す．

5） ページ管理ルーチンの機能構造

ページ管理ルーチン（page administrator）は，実記憶装置と補助記憶装置上のやりとりの管理をおこなう．ページ管理ルーチンの主要な働きは，ページ

ングに関する入出力オペレーションを開始させることであり，そのためのオペレーションを最適化することである．ページングの最適化には，**ページ・スロット分類アルゴリズム**（page slot sorting algorithm）を適用している．

ページ入出力開始ルーチン（図9-76参照）は，入出力チャネル・プログラムを作りあげ，ページ入出力オペレーションを開始させる．ページ装置の効果的利用をはかるため，チャネル・プログラムを連鎖し，入力と出力オペレーションを巧みに組合わすことによって，オペレーションの向上をはかっている．つまり，シーク時間の減少と回転待時間の最少化おをこなっている．このために，ページ入出力開始ルーチンは，スロット分類アルゴリズムを使っているのである．

これまで説明してきたように，補助記憶装置ページ・セットに用いられる装置は，グループ番号とスロット番号で参照できるような形式にされている．グループは１つまたは複数個のトラックから構成され，いくつかのページ・サイズのレコードが収容されている．これらレコードがスロットであり，１からNまでの番号がつけられている．このNは装置によって定まる番号である．

固定ヘッドの場合，スロット（$n+1$）はスロットnの処理のあと，同一回転内で処理できる．一方，移動ヘッドの場合だと，同一シリンダ内であれば，グループのいかんを問わず，同一回転内でスロット（$n+1$）はスロットnの処理につづいてアクセスできる．したがって，スロット分類アルゴリズムの出発点は，グループ番号のいかんを問わず，次の番号のスロットが同一回転内で処理できるということにある．このアリゴリズムでは，次の３つの要素が基本となっている．まずは，この３つにふれたあとで，スロット分類アルゴリズムを説明してゆく．

- **スロット待行列**（slot queue）：特定装置上の同一スロット番号のチャネル・プログラムの待行列
- **主スロット**（primary slot）：最後に割り当てられたスロット番号のつぎのスロット番号．つまり，スロット番号nの主スロットはスロット（$n+1$）ということになる．正しくは，グループ内のスロット番号Nの**モジュ**

ロー (modulo) である．したがって，具体的にはつぎのようになる：

$N=7$，$n=3$ のとき，主スロット＝4

$N=7$，$n=7$ のとき，主スロット＝1

- **代替スロット** (alternate slot)：主スロットの位置に最も近いスロット．代替スロットは，グループが2つ以上のトラックで構成されているときに存在する．**図 9-81** において，スロット1の主スロットはスロット2であり，スロット1の代替スロットは6である．

以下で，スロット分類アルゴリズムの処理手順をみてゆこう．

① ステップ1：スロット待行列の作成

チャネル・プログラムが利用可能だと，ページ枠を割り当て，装置依存のチャネル・プログラムを作りあげ，優先順位にもとづいて該当スロット待行列に入れる．同一スロット待行列にある2つのチャネル・プログラムが同一優先順位をもっている場合には，READ が WRITE に優先するようになっている．なお，移動ヘッドの場合，シリンダ番号の低い方から高い方へ，同一シリンダ内ではトラック番号の低い方から高い方へ並べられる．

表 9-2 でスロット分類アルゴリズムの例を示している．この例で，上段に示しているページ・スロットの配列は，ページ入出力開始ルーチンに対する要求の順序とスロットとを表わしている．これをもとに，スロット待行列が作成されたとすると，**ステップ1** で示しているようにスロット番号別にスロット待行列が作りあげられる．

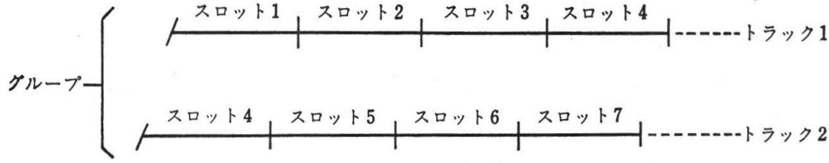


図 9-81　一次スロットと代替スロット

表 9-2　スロット分類アルゴリズムの例

《例》 ● ページ・スロット：1A, 2A, 1B, 3A, 4A, 7A, 6A, 1C, 3B, 5A, 5B ただし，1グループ7スロットで構成されるものとする． また，上記の表記で数字部分(1，2，…，7)はスロット番号，英字(A, B, C)はグループ番号を意味する	
ステップ1 (スロット待行列の作成)	スロット番号：　1　2　3　4　5　6　7 1：1A, 1B, 1C 2：2A 3：3A, 3B 4：4A 5：5A, 5B 6：6A 7：7A
ステップ2 (チャネル・プログラムの作成)	1A　2A　3A　4A(PCI)　5A　6A　7A　1B(PCI) 代替スロットは，スロット番号＋3のモジュロー7とする．スロットIBの主(スロット2)および代替(スロット4)共に存在しないので，上記連鎖でEXCPが開始する．
ステップ3 (PCI割込み)	1) 4AでPCI チャネル・プログラム1A, 2A, 3Aがポストされ，解放される．
	2) 1BでPCI チャネル・プログラム4A, 5A, 6A, 7Aがポストされ，解放される(1Bが最後のチャネル・プログラムなので，次のチャネル・プログラム，すなわち最も近い3Bが連鎖される．チャネル・プログラムは「1B　3B(PCI)」となる)．
	3) 3BでPCI 1Bがポストされ，解放される(5Bが最も近いチャネル・プログラムムなので，3Bに連鎖される．さらに，5Bの代替スロット1Cが連鎖される．チャネル・プログラムは「3B　5B　1C(PCI)」となる)．
	4) 1CでPCI 3Bおよび5Bがポストされ，解放される．スロット待行列にはチャネル・プログラムが存在しないのでもはや連鎖は行なわれない．
ステップ4 (チャネル終了割込み)	1Cでチャネル終了 (1Cがポストされ，解放されて，スロット待行列にはチャネル・プログラムがないのでリスタートは実行されない)．

② ステップ2：EXCP / チャネル・プログラムの連鎖

まず，該当するページング装置が実行中かどうか調べる．実行中でない場合だと，スロットの数の一番多いスロット待行列が選択され，チャネル・プログラム連鎖の先頭に位置づけられる．ただし，移動ヘッドの場合には，最後に実行されたチャネル・プログラムが対象としたシリンダに最も近いチャネル・プログラムが最初の連鎖として選ばれる．続いて主スロットの待行列からチャネル・プログラムをとって連鎖する．この操作を主スロットがなくなってしまう

まで続ける．主スロットがなくなると，最後のスロットの代替スロットを処理する．さらに，代替スロットの主スロットが存在すれば同様に連鎖する．ただし，代替スロットの代替は連鎖しないでおく．このような処理をつづけて，主スロットも代替スロットもなくなると，連鎖の操作を打ち切る．連鎖の対象とならなかったスロットは，そのままスロット待行列においておく．そのあとで，最後のチャネル・プログラムに PCI ビットをセットする．また，チャネル・プログラムの4つ単位に PCI ビットをセットする．このあと，EXCP により該当ページング装置に対する入出力オペレーションを開始する．

この様子を示したのが，**表 9-2 のステップ 2** である．スロット番号1がスロットの数が一番多いので，1A が選択されチャネル・プログラム連鎖の最初にくる．あとは，主スロットがなくなるまで，主スロットのスロット待行列の中からチャネル・プログラムを選んできて連鎖する．この例では，スロット 1B の主スロット（スロット2）および代替スロット（スロット4）のいずれも存在しないので，そのまま EXCP を開始する．

該当するページング装置が入出力オペレーションを実行中であると，目下実行中のチャネル・プログラムの最後の連鎖のスロット番号を調べ，そのスロットの主スロット待行列からスロットを選んで連鎖の最後につける．このあとは，前述の「実行中でない場合」の基準にそって次々と連鎖してゆく．

③ ステップ3： PCI 割込み

PCI 割込みが発生し，かつ PCI 割込みをひき起した CCW に別のチャネル・プログラムが連鎖されている場合だと，すでに実行したチャネル・プログラムをポストし，解放する（**表 9-2 のステップ 3 の1**)を参照のこと)．

連鎖されていない CCW で PCI 割込みが発生すると，最も近い次のスロット待行列からチャネル・プログラムが選択され，目下のチャネル・プログラムの最後に連鎖される（表 9-2 の**ステップ 3 の2)**および**3)**を参照のこと)．

④ ステップ4： チャネル終了割込み

最後の CCW にチャネル・プログラムが連鎖されていない場合，そのチャネル・プログラムは，ステップ3の場合と同様にポストされ解放される．その

あと，実行されたチャネル・プログラムに最も近いチャネル・プログラムが選択されて，つぎの新しいチャネル・プログラムの連鎖が開始する．表9-2の例では，もはやスロット待行列にはチャネル・プログラムが存在しないので新たなチャネル・プログラムの連鎖は行なわれない．

以上みてきたようなスロット分類アルゴリズムを採用することによって，ページ管理ルーチンは，ページング入出力オペレーションの効率化をはかっているのである．

五．おわりに

この章では，仮想記憶システムを実現するにあたって必要となる概念的な機能構造を明らかにしたうえで，仮想記憶システムのためのハードウェア構造とソフトウェア構造を紹介した．ここでは，仮想記憶システムを実現するために必要なハードウェア，ソフトウェアについてまとめておこう．また，このシステムを導入することによって得られるであろう効果も合わせてまとめておく．

仮想記憶システム用に必要となる新しいハードウェア機構としては，まず**動的アドレス変換機構**がある．この機構は，自動的に仮想アドレスを実アドレスに変換する．変換は，セグメント・テーブル起点レジスタから始まる．セグメント・テーブルとページ・テーブルを用いた変換では若干の時間を要する．この変換時間を減少するために，連想配列レジスタないしは変換索引緩衝機構を用いる．通常のシステムに比べた場合のシステム効率の低下は，この機構を採用することによって無視できるほどになる．

仮想記憶システムでは，これらの新らしい機構に加えて，従来どおりの資源も使用する．セグメント・テーブルとページ・テーブルは実記憶装置に記憶される．それだけ，監視プログラム部分がふくらむことになる．単一仮想記憶システムでは，ただ1つのセグメント・テーブルとセグメントごとに1つのページ・テーブルとが割り当てられるので，実記憶装置の追加容量は最少におさえうる．一方，多重仮想システムでは，これよりも多くの実記憶装置を使うこ

とになる．仮想記憶装置ごとにセグメント・テーブルとページ・テーブルとが用意されるからである．どちらのシステムでも，テーブルのために必要となる実記憶装置は，使用する仮想記憶装置の大きさによって左右される．補助記憶装置のあるものは，外部ページ記憶装置に適用される．外部ページ記憶装置の容量も，やはり仮想記憶装置の大きさによって変わる．さらには，外部ページ記憶装置と実記憶装置との間でのページングのために入出力チャネルが用いられる．このように，仮想記憶システムでは，実記憶装置，補助記憶装置，チャネルなど従来のシステム資源をも活用したうえで実現されるのである．

仮想記憶システムには新しいソフトウェアも必要である．ページ不在の処理，実記憶装置と補助記憶装置を管理し，効率的なページング・オペレーションを実現するための機能が必要であり，これらはページ監視プログラムとして集大成される．

これらハードウェアとソフトウェアから得られた効果や可能性にはいくつかのものが考えられる．仮想記憶システムでは，実記憶装置は要求時にのみ使用される方式，いわゆる要求時ページングの手法で利用されるが，この結果，つぎのような利点が引き出せる：

1. プログラムは，その実行を行うのに必要な**実効ページ・セット**(working page set) だけを要求する．この結果，同一システムでより多くのジョブの開始と実行が期待できる．
2. 実記憶装置の断片化はほとんど除去される．
3. 実記憶装置の容量よりも大きいプログラムを，そのまま実行できる．従来のシステムでは，オーバレイ手法やプログラムの分割手法を採用しないかぎり，この種のプログラムは実行できなかった．
4. どんなプログラムも，一時点で必要となる容量の実記憶装置しか使わない．したがって，データ通信システムのようなプログラムではピーク時には大きな記憶域を占有するが，閑散時にはわずかの実記憶装置しか使わない．ニーズに見合った極めて柔軟性のある実記憶装置の利用が実現できる．
5. 仮想記憶システムを，別のもっと大きな実記憶装置をもったシステムの

バック・アップ用のシステムとして使うことができる．これは，従来のシステムではほとんど不可能に近いことであった．

一方，プログラマの立場からすると，仮想記憶システムは従来の静的再配置システムにあったアドレス空間の厳しい制約からプログラマを解放する．仮想記憶装置は実記憶装置よりもはるかに大きいので，問題解決のために十分大きなアドレス空間を用いることができる．これによって，プログラムのオーバレイをする必要はほとんどなくなる．オーバレイ・プログラムを作成する場合には，プログラムに必要な記憶装置のサイズ，記憶装置の内容あるいはブランチの仕方などについて前もって検討しておかなければならないというわずらわしさがある．それに，オーバレイを採用したプログラムを修正変更したり，保守したりするのは容易なことではない．仮想記憶システムでは，要求時ページングの手法を用いて，システムが実記憶装置を管理し，実記憶装置より大きなプログラムであっても，自動的なオーバレイを実現してくれる．したがって，プログラマは仮想記憶装置の中の連続した広いスペースを相手にプログラミングできる．この結果，プログラミングに要するわずらわしい手間や時間がはぶけ，プログラマには，与えられた問題解決にだけ専念できる環境条件が与えられることになる．

最後になったが，オペレーティング・システムの利用にあたっても，仮想記憶システムから利点が引き出せる．オペレーティング・システム自体が仮想記憶システムに収容され，その多くの部分がページに分割されていることから，従来のシステムと異って，必要なだけ常駐機能を拡げることができる．ページング可能な常駐機能は，必要なときにのみ実記憶装置を使用する．しかも，それらの機能は，いつでもシステムやユーザで共用できる．従来のシステムでは，常駐ルーチンは常時主記憶域に留まっていたため，この種のルーチンを極力制限する必要に迫られたのであるが，仮想記憶システムではこの心配はなくなる．

仮想記憶システムでは，主記憶装置を主体としたシステム資源の有効活用，プログラミングの制約解除，およびオペレーティング・システムの各種機能の効率的利用が期待しうるのである．

第 10 章

ゆるやかに結合された
複合システムの機能構造

一. 複合システムの定義と出現の背景

本章では，いわゆる“**ゆるやかに結合された複合システム**（loosely coupled multi-processing system：以下単に“**複合システム**”という）”を，IBMにおいて実現された**JES 3**（Job Entry Subsystem 3）に従って，簡潔に解説する．この種の複合システムは，従来の複合システム（tightly coupled multi-processing system）と同様，あるいはそれ以上に今後志向される巨大・複雑システムの代表的形態の1つといえる．また，JES 3 はジョブ・スケジューラの一種であるが，これまでのジョブ・スケジューラとは大幅に異っており，今後のジョブ・スケジューラの1つの方向を示唆しているといえる．なお，本章では**緊密に結合された複合システム**（tightly coupled multi-processing system）は追求しない．また，読者の理解を助けると思われるときは，複合システムの設計方針に立ち入ることもあるが，全体としては設計の立場よりも，利用者の立場から言及する．次のような事柄についてはは特にテーマをとりあげて議論せず，どうしても必要があるときだけ，付随的に説明することにする：

- 複合システムを実現する場合の包括的議論，および各方法の優劣
- 各方法の問題点
- JES 3 の性能（効率）

コンピュータの利用の範囲が拡大し，質的に高度化し複雑化してくると，一

台のコンピュータではその処理能力が限界に達してしまう．この事態に対処する最も素朴な解決策は，新しく2台目を設置して，不足分を吸収する方法である．こうした台数の増加にともなって，その運営管理が問題となってくる．通常は，最初に設置しシステムの運用技法を，新設のコンピュータにも踏襲する．たとえば，システム一台を5名で操作していれば，新たに5名を増員して，同様の運用方法を適用することである．このさいの問題点は，システムの最適化がこれで実現されるかどうかということである．もし，ジョブが2台に分割され，それぞれのシステムでの最適化が計られたとしても，処理は互に関係のないシステムで行われるので，つぎの問題が依然として残る：

- それぞれのコンピュータに振り分けるべきジョブの選択の妥当性
- 各々のコンピュータ資源が，一台ごとに最適化されていても，すべてのシステムを全体として見た時には，必ずしも最適化されているとは限らない．
- 入出力装置，あるいはそれに収容されるデータ・セットのシステム間での移動の問題

この種の諸問題を解決し，システムの**可用性**（availabilty）と**信頼性**（reliability）を向上させるための最良の方法は，適当な整合部を考えて，その整合面からみれば，利用者やオペレータが“あたかも唯一つのハードウェア・システム”であるかのごとき対処ができる環境を作り出すことである（**図10-1**参照）．このための複合システムは，つぎのような特徴を持つ．

複合システムは，2個以上の独立した単一処理装置（コンピュータ・システム）の集まりであり，各々の単一システムは互に情報の通信（交換）ができる．ここでいう単一システムとは，主記憶装置を2個以上のCPUが共通するシステム（tightly coupled multi-processing system）であってもかまわない．“ゆるやかに結合された複合システム”の通信とは，各単一システムが共通にアクセスできるDASDをもつこと，および単一システムの間でCPU間の直接の連絡ができることを意味する．また，これらの通信の制御はソフトウェアで行われる．

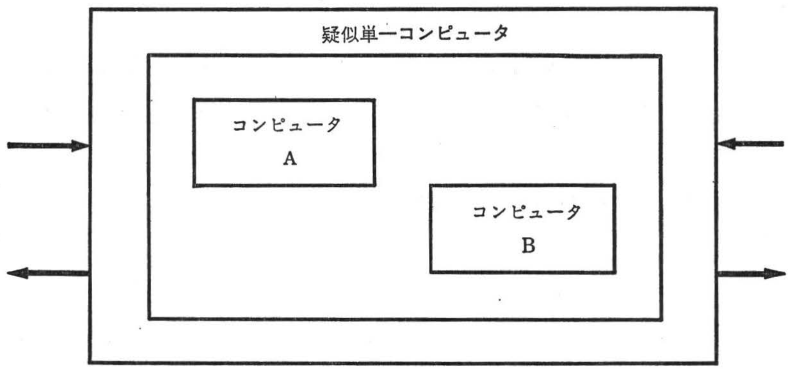


図 10-1　疑似単一コンピュータの構想

複合システムは，利用者の観点からみれば，あたかも大きな単一システム，つまり**擬似単一システム**（pseudo single system）である．このように**“ゆるやかに結合された複合システム”**（loosely coupled system）が実現されたときの利害関係は，ユーザー部門やシステム設計者にとってみれば，所要の結果が得られるかぎりにおいて無関心であり，プログラマからすれば従前のプログラムが支障なく実行できれば無関心となる．一方，オペレーションの担当者や管理者からすれば，オペレータ数が減少でき，オペレーションが容易で確実となれば歓迎することになろう．したがって，直接の利害関係者はオペレーション部門ということになる．そこで，複合システムにはつぎのような設計目標を定める必要がある．これは，一言でいえばオペレーション担当者からみての擬似単一システムである：

- 単一のジョブ待行列（single job queue）
- 入出力装置は，各単一処理システムに接続されているというよりも，擬似システムとして同種類のものを一まとまりとして管理すること（I/O device pool）
- 擬似総合コンソールを設定し，傘下の各システムを総合コンソールから制御できること．

二義的には，つぎのような目標が考えられる：

- 資源の割振りについては，複合システム全体としてみること．たとえば，単一システムでは，主記憶域の大きさが十分ないためスケジュールできないプログラムが，複合システムではスケジュール可能かも知れない．
- フェールソフト―どれかの単一システムで障害が起ったとき，健全な代替単一システムで代行できるかも知れない．
- 最適スケジューリング―ジョブ待行列の中から実行すべきジョブを選択するときには，その優先順位だけでなく，資源利用の最適化を考慮できるようにすること．

また，つぎに述べるような目標は，複合システムでジョブ処理を行う場合に，必然的にその解決策を組み込んでおかねばならない種類のものである：

- 互に依存関係のあるジョブの実行順序の制御，依存関係とは，あるジョブの実行が，別のジョブ（1つ以上）の実行終了後しか開始できないような関係である．つまり，依存ジョブ処理が正しく遂行される機能が必要となる．
- 特定時までにジョブを開始することの制御（もっと望ましくは，特定時までにジョブを終了すること）．

上記の2つの条件は，独立した単一システム群で処理をしている場合は，オペレータが制御していたものである．複合システムにおいては，一般にジョブの待行列が大きくなり，かつどのジョブがどのコンピュータ・システムで処理されるのかを予測できないので，オペレータが制御することは，かなり困難になる．したがって，システム自身に面倒をみさせる必要がある．

二．複合システムの構成

1.　ハードウェア構成

単一のジョブ待行列を処理してゆくためには，複合システムの中の一台のコ

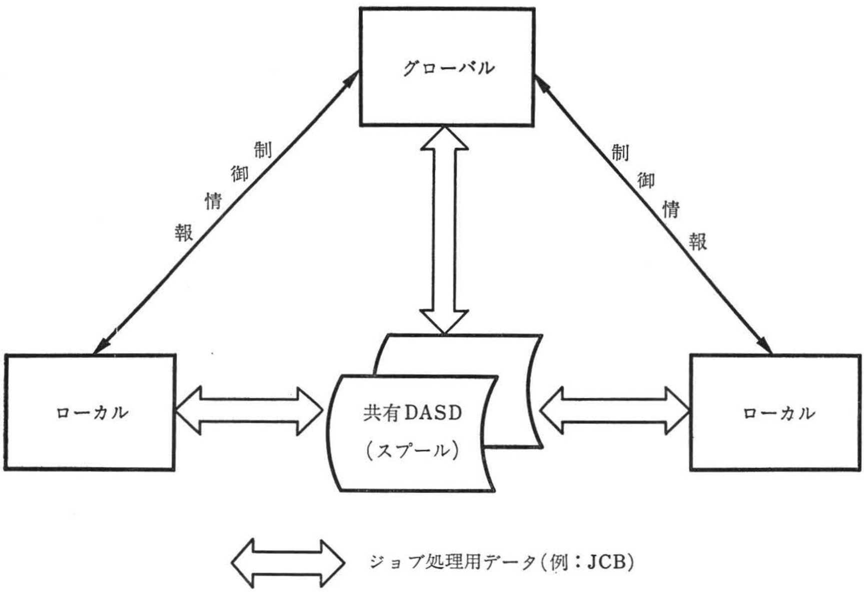


図 10-2　複合システムの基本構成

ンピュータはジョブ管理を行う機能が必要である．このコンピュータを**グローバル・システム**（global system）という．残りのジョブ処理用のシステムは，**ローカル・システム**（local system）と名づける（**図 10-2** 参照）．

各ローカルは，**チャネル対チャネル付加装置**（channel to channel adapter）を通じてグローバルと通信を行う．この通信は，グローバルからローカルへの指示，およびローカルからグローバルへのお伺いのための制御情報の交換のために使用される．また，すべてのシステムは DASD（スプール）を共有する．これは，入力されたジョブ情報を保持し，必要に応じてグローバルおよびローカル・システムから参照される．

ある特定のコンピュータ・システムをグローバルとして，ジョブの待行列の管理に専念させることには，つぎのような利害がある：

- この種の管理に関しては，特定的である（この機能のための処理プログラムは一個だけでよい）．他のすべてのコンピュータ・システムは，ジョブ

待行列に関する情報に関しては，特定システムに依存するだけでこと足りる.

- 特定コンピュータに故障が生じた場合には，その代替がむずかしい.
- 他のローカル・コンピュータから，ジョブ待行列処理の負荷を除去できる.

1）グローバルの機能

グローバル・システムは，複合システムの中核である. 故障時の対策については後述するので，ここでは正常時の機能だけを考えてみよう：

① 複合システムに対するすべてのジョブ入力は，グローバル・システムに接続する入力装置から読み取られる. 想定される入力装置としては，カード読取装置，磁気テープ装置，磁気ディスク装置，RJE 端末装置，タイム・シェアリング端末装置，コンピュータ・ネットワーク用コンピュータ，擬似カード読取装置などがある.

② ジョブがローカルで実行開始されるまで（特別な場合には，グローバルで実行される場合もあるが）の前準備（たとえば JCL の解釈）および実行時の**システム出力**（system output）の処理を行う.

③ ジョブが入力されてから，システム外に追放されるまでの諸段階の制御. ジョブの実行時における介入については，必要に応じておこなう.

④ 上述②および③に必要なすべての資源の取得・管理，およびその開放の面倒をみる.

⑤ ローカル・システムからの情報要求についての応答

⑥ 上記①から⑤を実現するために，システム状況を常時把握する.

2）ローカルの機能

ローカルの機能は，上述のグローバルの機能で明らかなごとく，グローバルに対して受動的・従属的である. ローカルは，グローバルの指示に従って，グローバルがあらかじめ用意してくれたジョブを実行するだけである. 単に，特定ジョブの実行だけに専念するのが，ローカル・システムである. したがっ

て，下記のような機能は，通常のシステムとは異なり，すべてグローバルが面倒をみてくれるので，ローカルはこれらを行わない：

- 入力ジョブの読取り
- JCL の解釈
- 次に実行すべきジョブの選択
- 必要な入出力装置の割当て
- 印刷・穿孔などの周辺操作
- 現在どんなジョブが実行されているかの認識
- ジョブ実行後の後始末など

これまでふれたハードウェア構成で推測されるように，この種の複合システムは，一種の**ハイアラキー・システム**（hierarchy system）である．したがって，グローバルとローカルの間では，何らかの意味での相互の通信が必要となる．相互の通信内容としては，**制御情報**（control information）と**処理情報**（processing information）とが必要である．制御情報とは，たとえば，ローカル・システムがグローバル・システムに「システムが空いたので，ジョブを下さい」と依頼することであり，チャネル対チャネル付加装置を通じて行う．一方，処理情報とは，グローバル・システムがローカルの要求を受理し，ローカルがジョブ処理を開始するに当って必要とする情報を受渡すことであり，共有 DASD（スプール）を通じて行う．チャネル対チャネル付加装置は，転送速度はきわめて高速であるが，実際に転送される制御情報は少ない．共有DASD装置は，それほど高速ではないが，転送される情報量は多い．このような二本立てのハードウェア構造を採用したのは，その用途に見合う選択と代替を実現するためである．

2. ソフトウェアの構成

ある単一システムが複合システムに成長するのは，単一システムだけでは負荷がこなせなくなるからである．複合システムへの移行を円滑に行うためには，現実の稼動システムにハードウェア上でチャネル対チャネル付加装置とス

プール用 DASD を追加しただけで，複合システムが運用できるように，簡単に移行できないと困る．つまり，複合システムを動かすソフトウェアは，単一システムのソフトウェアを含み，かつ何かの都合でそうしなければならないときは単一システムの環境での運転も可能でなければならない．

したがって，複合システムのソフトウェアの要件としては，つぎの諸要素が加味されていることが前提となる：

① 従来のオペレーティング・システムの機能をことごとく含むこと．

② さらに，グローバルの処理機能が追加されること．グローバルの機能は，従来のジョブ管理に相当するものである．

③ 追加の具体化としては，伝統的なものへの直接の組み込みと拡大，または，適当な**境界**（interface）を定義し，追加の部分を独立して組み込む方法とがある．複合システムでは追加部分が厖大であること，将来の独立した新しいタイプの追加を可能にするには，後者を採用した方が得策である．

④ 独立した追加を可能にするには，既存システムとの境界の定義と両方向での情報のやりとりの標準的方法を確立することが必要である．

⑤ 追加される機能は，前述のようにジョブ管理に属するものであるので，新ジョブ管理機能と命名してもよいのであるが，IBM ではジョブ・エントリ・サブシステム（job entry subsystem）としている．ただし，この名前はエントリ（entry）を強調すると大変あやまった印象を与える．単にエントリのためにどうして厖大なコーディングと専用プロセッサが必要かなどの反問を食いそうである．すでに述べたように，ジョブの入力（entry）一実行一出力とそのジョブの追放のすべての面倒をみるのが，ジョブ・エントリ・サブシステムである．

以下では，IBM の **JES 3**（Job Entry Subsystem 3）にもとづいて，そのソフトウェア構成を追求してゆく．

1）ジョブ・セグメント

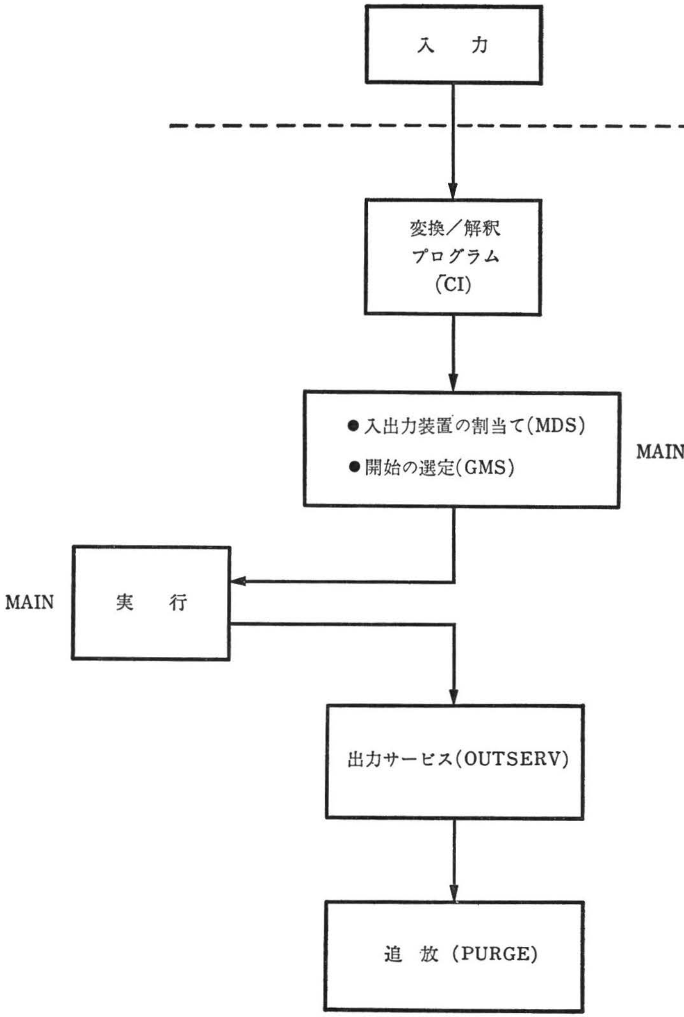


図 10-3 ジョブ・セグメント

伝統的なオペレーティング・システムでのジョブの処理過程は，ジョブの入力，JCL の解釈，諸資源の割当て，実行，ジョブの出力と解放である．ただし，この過程は，きわめて暗示的に遂行された．プログラマやオペレータが自由にこの過程に介入する余地はなかった．JES 3 においても，この過程は，本質的には同様であるが，すでにふれたようにグローバル・システムがすべての

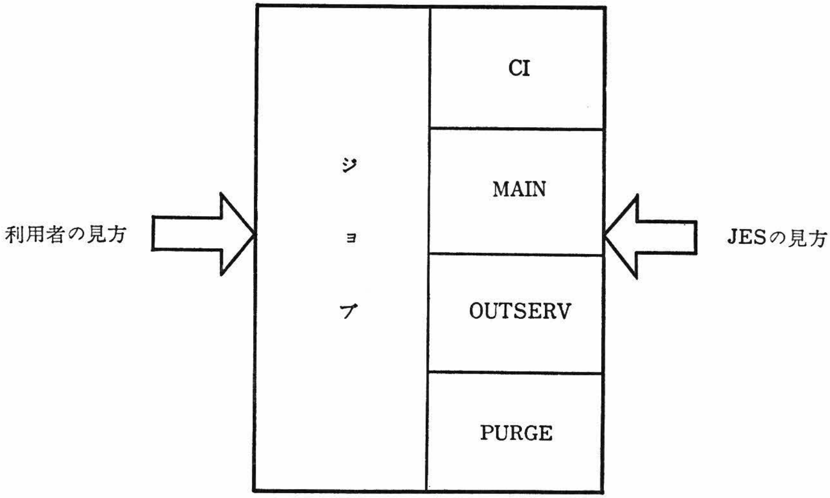


図 10-4　利用者と JES の見方

前準備，および後始末を行うので，この処理過程は明示的に取り扱われる．したがって，この各処理過程を特に**ジョブ・セグメント**（job segment）という．通常のジョブは，**図 10-3** で示したようなジョブ・セグメントに分割される．

すなわち，入力されたジョブはすべて，CI-MAIN-OUTSERV-PURGE の4つのジョブ・セグメントから構成される．これを**標準的ジョブ**（standard job）という．明示的であるというのは，ジョブ制御情報の中に，上記のジョブ・セグメントが含まれ，JES 3 は**図 10-4** で図示した順序でセグメントを取り出して処理するということである．このようにセグメントを明示的に定義すると，CI-OUTSERV-PURGE のように一部セグメントが欠落したジョブ，あるいは追加のセグメントをもったジョブの処理も可能となる．このように標準以外のセグメント（と順序）をもつジョブを**非標準ジョブ**（non-standard job）という．

2）ジョブ・セグメントを処理するためのプログラム（DSP）

各ジョブ・セグメントを処理するためのプログラムを，**ダイナミック・サポート・プログラム**（Dynamic Support Program；略して **DSP**）という．ただ

し，DSP は単にセグメント処理だけに関係するのではない．ジョブ入力の段階もこの DSP で処理される．DSP という名称の由来は，各ジョブ・セグメントの DSP 処理が必要になったとき，かつその時のみ，動的にロードされ実行されるからである．

標準ジョブのための DSP は，当然のことながら IBM より提供される．追加的な非標準ジョブに対しては，ユーザが作成し，ライブラリに収納しておくことが必要である．前述のグローバル・システムの説明で推察がつくことであるが，これらの DSP は，ジョブの実行段階を除いて，すべてグローバル上でのみ機能する．

3）ジョブ・セグメント・スケジューラ（JSS）

入力ジョブの読取りは，JES 3 を通じて自動的に行われる．この段階で，標準ジョブについては，CI-MAIN-OUTSERV-PURGE の 4 つのセグメントが生成され，ジョブ制御情報としてスプール上に優先順位にしたがって収納される．この制御情報が連鎖されたものを**ジョブ制御テーブル**（Job Control Table; 略して **JCT**）といい，各ジョブに対応する**記入項目**（entry）から構成されている．この JCT は，従来のジョブ待行列を代替するものである．

一定数のジョブの読込みが終ると，制御権は**ジョブ・セグメント・スケジューラ**（Job Segment Scheduler；略して **JSS**）に移る．JSS は，JCT の最初の記入項目，つまり最も優先順位の高いジョブの記入項目を取り出して，セグメントを検査する．セグメントは，“処理中”，“終了” および “再試行” の標識を持ち，何も標識がついていないときは未処理を意味する．セグメントが未処理であれば，そのセグメントをスケジュールする．すなわち，JSS は対応する DSP を実行するのに必要なシステム資源（主記憶装置，入出力装置，データセットなど）を割当て，最終的には該当する DSP をロードし，制御権を渡す．

いったん制御権が DSP に移ってからの進行制御は，後述する**多重機能モニタ**（Multi-Function Monitor；略して **MFM**）が行う．ともあれ，DSP の実行が終了すると，制御権は再び JSS に戻り，セグメントの後始末（使用システ

ム資源の解放，セグメント処理終了の標識のセットなど）を行い，最終的にはMFM に制御権を渡す．

興味あることであるか，JSS もまた DSP である．

4）多重機能モニタ（MFM）

複合システムの JCT の各記入項目に対する同時点でのサービスの種類は多様である．たとえば，ジョブAは CI，ジョブBは MAIN（入出力装置の割当て），ジョブCは MAIN（開始の選択），ジョブDは MAIN（実行）といった具合いである．このように異ったジョブの異ったセグメントに対するサービスを効率的に行うために，JES 3 は複数個の DSP を同時に走行させるための機能をもっている．これは，本質的にはオペレーティング・システムの多重タスキングである．この機能を遂行するための要素は，前述のジョブ・セグメント，DSP，JSS，および MFM である．

MFM は，オペレーティング・システムのタスク・ディスパッチャに相当する．MFM の動作はつぎの通りである．つまり，この MFM がいったん制御権を受けとると，**機能制御テーブル**（function control table）を検討する．もし，該当記入項目がサービス待ちの状態であると，そのセグメントを処理すべき DSP に直ちに制御権を渡す．

機能制御テーブルは，各セグメントが JSS によってスケジュールされたときに作成される．前出のジョブ制御テーブルが外面的であるのに対し，機能制御テーブルは内面的である．このテーブルは，ディスパッチングのための情報を MFM に提供する．そのために，テーブルの各記入項目はあらかじめ定義された優先順位にしたがって配列される．なお，セグメントに対するサービスが終了した時点，つまり DSP の実行が終了すると，該当記入項目は JSS によって消去される．

各ジョブに必要な DSP のサービスを提供するために，対応する DSP に制御権を受け渡すのが，MFM の機能であるが，DSP の割り振りは DSP の指名順位にもとづいて行われる．一たん DSP に制御権が移ってから，数ミリ秒以上その DSP で制御権を保有することは，他の DSP 利用権をいちじるしく阻

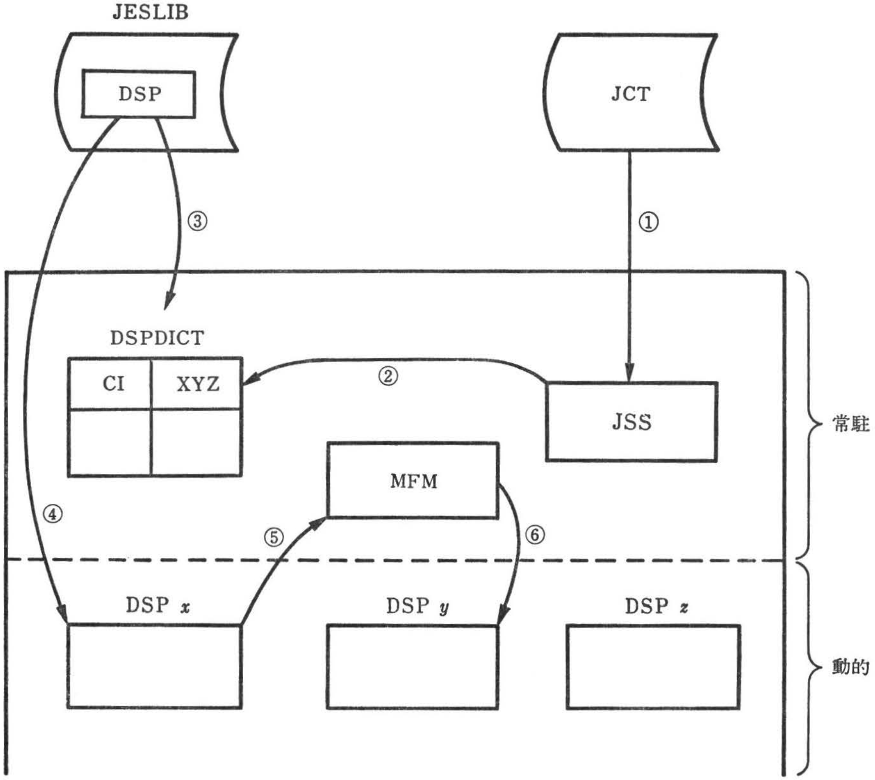


図 10-5　DSP の構成

害するので，適当なタイミングで自主的に制御権を戻すように作成しておかなければならない．

以上のソフトウェアの関係をまとめると，**図 10-5** のようになる．

3.　コンソール

複合システムの構成の中で重要な要素の１つとして，コンソール・サービスがある．JES 3 では，この機能のために総合コンソールと機能コンソールとが利用できる．

擬似単一システムのすべての制御を単一のコンソールから行うのが，総合コンソールである．すなわち，総合コンソールは，すべてのシステムに対して

（または，すべてのシステムから），オペレーティング・システム指令の送付および返信の受信，JES 3 の指令の送付および返信の受信，オペレーティング・システムおよび JES 3 のシステム状況の把握と変更などに適用される．

一方，特定の機能に関しての指令や返信については，その機能に関係する**コンソール**を指定して，それのみに限った方がよい場合がある．このために用いられるのが機能コンソールである．たとえば，磁気テープ装置のプールに対するコンソールを設定して，テープ操作にかかわる指示・応答はすべてこのコンソールを通じて行う．この方が，オペレータの配置が円滑に行え，オペレーション効率の向上が期待できる．機能コンソールは，ユーザの指定によってかなり自由に選択できる．一例としては，DASD 用コンソール，磁気テープ装置用コンソール，ジョブ入力用コンソール，システム出力用コンソール，エラー用コンソール，RJE 用コンソールなどが考えられる．

このように適当にコンソール・システムを設計すると，オペレーティング・システムのコンソールは，ほとんど不要となる．ただし，現在のところ，オペレーティング・システムとの関連で，IPL と **DSI**（Dynamic System Interchange：走行中のグローバルを交換すること）だけはオペレーティング・システムのコンソールを通して行われる．

4.　スプール・データ管理

入力ジョブのカード・イメージおよびシステム入力（SYSIN）を収容するために，それぞれのシステムで共用される DASD が設定される．これが，スプール用 DASD である．スプールには，この他に JES 3 の内部処理で用いられる制御情報（たとえば，JCT）を収容する．

スプール上では，異ったデータ（レコード）が異った時点で参照されるために，通常のオペレーティング・システムとは異った即目的的なデータ管理が必要となる．このデータ管理体系を一般に**スプール・データ管理**（SPOOL data management）という．このデータ管理の内訳はつぎの通りである：

● **データ・レコード**：JCL あるいは SYSIN のように，複数個の種類のレコ

ードが一まとまりとして取り扱われるもの (multi-record file) と，参照が単一形式のレコードに限られるようなもの (single record file)

- **トラック・グループ**：ボリュームを適当なトラックのグループに分けて，各々の multi-record file と single record file の収容の効率化と，アクセスのための時間節約をはかる．
- **スプール・レコードのためのバッファリング**
- **入出力オペレーション**：スプール・レコードの入出力は，特定的であるので，一般的には入出力監視プログラムの処理は冗漫といえる．そこで，JES 3 スプール・データ管理は，入出力監視プログラムに対してドライバ（駆動部）として機能する．
- **SYSIN/SYSOUT の仲介**：問題プログラムの入出力命令 (GET/PUT, READ/WRITE) は，スプール・データ管理に適合するように実行時に仲介される．なお，SYSIN，SYSOUT のブロック化因数 (blocking factor) は無視され，ブロック・サイズは JES 3 の標準に従う．

5. オペレーティング・システムとの整合

JES 3 は，オペレーティング・システムの基本的要素として当初から設計されたというよりも，従来のオペレーティング・システムのジョブ管理プログラムの改良として誕生したいきさつがある．しかしながら，その規模の厖大なことから，サブシステムと呼ばれている．この種のオペレーィィング・システムの増・改築時の注意点として，既存システムからの機能の移管に伴う既存システムの修正，および既存システムが移管したサービス（および JES 3 の特有サービス）に対する JES 3 への要求方式の標準化がある．このような整合部のことを**サブシステム整合** (subsystem integration) という．たとえば，従来では，ジョブの開始は，**開始プログラム** (initiator) が行っていたが，JES 3 ではつぎのような順序となる：

既存システム (開始プログラム)		整合部		JES 3
次に実行すべきジョブを下さい.	──→	要求をJES 3 の型に変換	──→	ジョブ待行列の中から，実行すべき“最良”のジョブを選

JES 3 から提供された情報にしたがって必要なシステム資源を割当てて，実行すべきプログラムに制御権を移す.	←	開始プログラムに準備ができたことを知せる	←	んで，実行に必要な情報を渡す.

整合部は必要悪ではあるが，一度このような標準的な方法を確立しておけば，異ったサブシステムが設計された時に，容易に統合することができるという利点がある．なお，現在の JES 3 では34種の機能（要求コード）が設定されている．

三．JES 3 の処理機能

JES 3 よにって読み取られたジョブが，実行前までに行われる準備，実行後に行われる後処理については，前節で概観した．つぎのような各段階での処理がある：

- 変数 1 解釈（Converter/Interpreter；略して **CI**）
- 主処理（main）
  - **主装置スケジューリング**（Main Device Scheduling；略して **MDS**）
  - **一般化された主スケジューリング**（Generalized Main Scheduling；略して **GMS**）
  - 主処理（main）
- 出力サービス（output service）
- 追放（purge）

各段階の処理は，DSP で行われるので，その説明はプログラムの機能説明でもある．これから記述するのは，IBM 提供の DSP による処理であるが，すでにふれたようにユーザがそれを変更あるいは置換することにより，新しい機能を得ることは容易である．一般的にいえば，以下の機能は従来のオペレーティング・システムに対する拡張である．

1. 変換／解釈の機能（整合部）

入力されたジョブは，一時共有DASD（スプール）に貯えられる．そのあとで，JCL の変換／解釈が行われる．JES 3 の**変換／解釈プログラム**（converter/interpreter）は，駆動機能が主体で，実際の変換／解釈はオペレーティング・システムの**解釈プログラム**（interpreter）が面倒をみる．このプログラムで抽出された制御情報は後段の処理にそなえて，スプール上に保持される．なお，変換／解釈プログラムの主要な機能は，つぎのものに集約できる：

- 同時にサービスを受けるジョブの個数の制御
- 頻繁に利用されるプロシージャの常駐化
- 利用プロシージャ・ライブラリの利用
- データ・セットのカタログの動的割振り，および更新制御
- 変換／解釈用パラメータの指定
- 共通データ・セット名の探索の省略
- 処理中のエグジットの設定

2. 主処理（MAIN）の機能

この処理は，処理の時間的経過にもとづいて，つぎの 3 段階に大別される：

① **主装置スケジューリング**（**MDS**）：ジョブが必要とする入出力装置の割当てを行う．

② **一般化主スケジューリング**（**GMS**）：特定のプロセッサ（コンピュータ・システム）に対し，つぎに実行すべき“最良”のジョブを選び出す．

③ **主処理**（**main**）：ジョブの実行中のサービス

以下で，それぞれの機能を説明してゆく：

1）主装置スケジューリング（MDS）の機能

変換／解釈が終了したジョブは，MDS の中の**割当て待行列**（allocation queue）に置かれる．MFM が MDS をディスパッチすると，MDS は待行列の

中の最初のジョブを取り出して，入出力装置を割当てる．この段階で，必要があれば，磁気テープや磁気ディスク・パックなどのボリュームを機械室に運び込むよう指示するためのメッセージを機能コンソール上に打ち出す．割当ての済んだジョブは，**マウント待行列**（mount queue）におかれ，つぎにディスパッチされたときに，オペレーターに対してマウント指示を出す．マウントが済んだことを確認したあとで，そのジョブは GMS 待行列に移される．

割当てのさい，必要入出力装置が不足する場合は，該当ジョブは次の機会にまわされる．

JES 3 の装置割当ての方法にはつぎの 3 種類がある：

① 全ジョブ・ステップで必要とする装置の最大を確保する．

② ジョブに必要な最小の入出力装置を確保する．

③ ユーザが指定した以外の入出力装置に関しては，JES 3 に迷惑をかけない．

いずれの方法も，根本的には全ジョブ・・ステップに対して，かつジョブの実行前に割り当てるという考え方である．従来は，ジョブ・ステップが終了するつど，開始／終止プログラムが前のステップの不要となった入出力装置を解放し，しかるのちにあらためてつぎのステップのための入出力装置を割り当てる方法であった．

JES 3 においては，ジョブの実行が終ったあとに，再び MDS が要請され，**ディスマウント待行列**（dismount queue）から終了ジョブが取り出され，システム資源に返される．

MDS の主要な機能をまとめると，つぎのようになる：

- 3 種類の入出力装置割当ての セット・アップ（すべてオペレーティング・システムにまかせることも含む）
- ボリュームの取り出し指示（たとえば，ライブラリにある機能コンソール上に）
- ボリュームのマウント時のエラー試行の制限（保全のため）

- 入出力装置の早期解放（ジョブ／高水準セット・アップ）
- 入出力装置のスケジューリング段階へ入るときの手動モード
- ディスマントされたボリュームの後始末の指示(「ライブラリに返却せよ」とか，後で再び使用する予定があるので「装置のそばに置いておけ」とかの指示）
- 共通装置群への共通メッセージ（機能コンソールを使って）
- ボリュームが見当らないときの懸案処理
- 特定ジョブに対する専用指定の処理
- 装置選出時の探索順序指定の処理
- JES 3 による（オペレーティング・システムの）マウント指令の代行

2）一般化主スケジューリング（GMS）の機能

MDS の指示にもとづいて，すべての入出力装置に所要のボリュームがマウントされると，該当ジョブは GMS の**選出待行列**（selection queue）に移される．オペレーティング・システムの**開始プログラム**（initiator）から，サブシステム・インターフェースを通じてジョブの依頼がくると，GMS は選出待行列の中から“最良”と思われるジョブを選んで開始プログラムに渡す（**図10-6**参照）．これ以降は，GMS の制御を離れ，MAIN サービスの領域となる．

GMS が“最良”のジョブを選択するさいの基準となるパラメータとしては，つぎのものがある：

① システム側の資源条件

- 実行中のジョブ混合比（入出力対 CPU）
- ジョブ・クラス（グループ）の混合（同じ性質をもつジョブ数の制限．JES 3 では，クラスの概念は従来のシステムと異なり，明示的である．また，いくつかのクラスをグループ化することができる．資源の割当ては，このグループに対して行われる．従来のシステムのクラスを JES 3 の立場でみれば，クラスとグループとが一致している特別な場合である）
- 空き記憶域，あるいは**実効ページ・セット**（working set）の大きさ

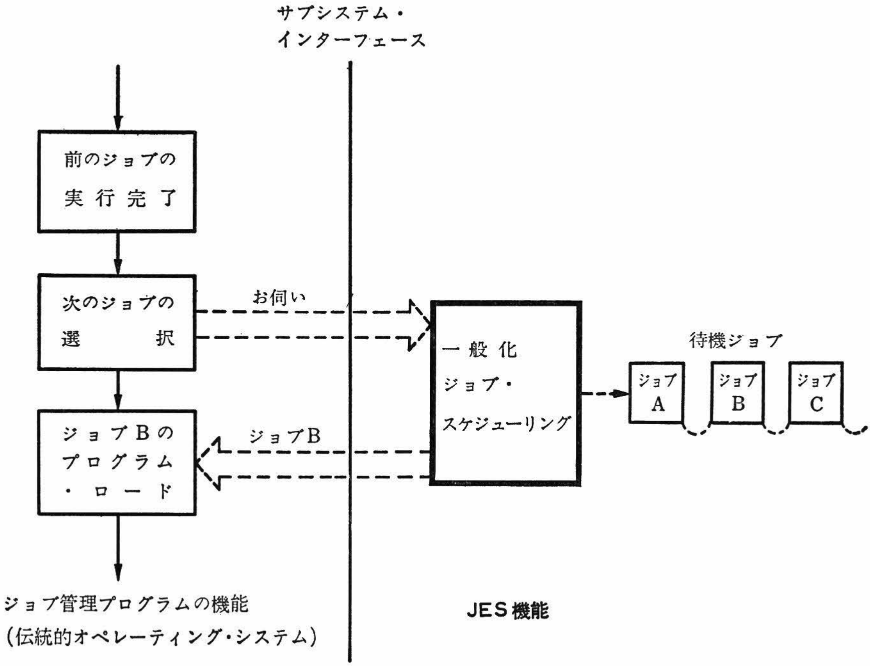


図 10-6　ジョブ管理プログラムと JES との相互関係

- ユーザの CPU 指定への合致

② **ジョブ側の条件**

- ジョブ優先順位
- 特定プロセッサ（コンピュータ・システム）の好み

③ JES 3 (GMS) 側の条件（たとえば，シフトごとに変更が可能）

- 最初のジョブ（優先順位順）
- 最良のジョブ混合
- 空きの記憶域

3）主処理（main）の機能

いったんジョブがローカル・システムに渡されると，JES 3 はジョブの実行

中はほとんどサービスを提供しない．というよりは，この段階ではJES 3 のサービスの必要がないといってよい．内部的には，GET/PUT のような命令は，その時点でJES 3 のルーチンにとび込み，スプール入出力形式に適合するように変換され，スプール・データ管理の機能を通じて入出力オペレーションが遂行される．この段階でのJES 3 の関与は，出力量を測定して，限界値を超えたときに警告を出すとか，打切りなどの指示を行うといった程度である．また，副次的な役割りとしては，JES 3 の初期値化の段階で，各ローカルとグローバルとが**握手する**（hand-shaking）時に機能する．

4）追加機能

これまでみてきたように，主処理の段階ではジョブは内部的に様々の待行列（割当て待行列，マウント待行列，GMS 待行列，選出待行列，ディスマウント待行列）を通過する．この待行列の中のジョブの制御に関しては，つぎの2種類の方法があり，ジョブの差別待遇を可能にする：

① 優先順位コントロール：MDS，GMS 待行列中にあって適当な時間（厳密にいえば，探索空通過回数）経過したジョブは，その優先順位を JES 3 がくりあげる．また，一定以上の優先順位に達しないと，探索の対象としない．MDS が終了したものは，さらに優先順位をあげることもできる．

② ジョブ数制限：各々のプロセッサに対して，実行可能な特定クラス（グループ）のジョブ数を制限する．この方法は，複合システムとしても定義することができる．また，実行ジョブ数のみならず，セット・アップ・ジョブの数にも同様に適用することができる．

3. 出力サービスの機能

ジョブ実行中の SYSOUT は，スプールに書き出される．すべてのジョブ・ステップの終了後，これらのレコードは，出力サービスによって印刷，穿孔，端末送信などがおこなわれる．これは，従来の**出力書き出しプログラム**（output writer）に相当し，**熱い書き出しプログラム**（hot writer）と**冷い書き出しプログラム**（cold writer）とがある．熱い書き出しプログラムは，オペレータの介

入なしに，書き出しデータが利用できるようになったとき自動的に書き出しが開始される仕組みになっているものである．

4. 追放プログラムの機能

この段階は，ユーザとしては無関心であっていいものである．**追放プログラム**（purge）の必要性は内部的であり，これまで述べてきたように，ジョブあるいはジョブ・セグメントを明示的に定義し，そのセグメントの処理に必要となるシステム資源の確保を明示的に行ったために，このような段階が必要となる．追放プログラムの主要な機能は，ジョブの実行のために確保されたシステム資源の中で未解放のもの（たとえば，スプール・スペース）を解放し，最終的な**会計データ**（accounting data）を **SMF**（System Management Facility）に書き出すことである．

このステップが終了したジョブは，もはや複合システムには存在しない．たとえば，オペレータがシステムにジョブの状態を照合すると，「こんなジョブは存在しない」(「該当ジョブはすでに終了した」ではない）という返答がかえされるのである．

5. 機能指定と制御

これまで説明してきた諸機能のほとんどすべてを，外部的に指定することができる．この段階を初期化という．初期化された複合システムの各指定（パラメータ）を変更し，あるいは潜在的なものを顕在化する方法としては，JES 3 制御ステートメントを通じて行うやり方と，コンソールを用いて行うやり方とが存在する．

1）初　期　化

JES 3 の諸機能は，ユーザのオペレーション環境，意図などにもとづいて，選択可能である．これまで説明してきた**4 段階**(CI-MAIN-OUTSERV-PURGE) の諸機能，スプール関係情報，入出力装置の接続関係，および複合システムの構成などはすべて関係項目ごとに JES 3 に指示される．代表的なものは，MA

INPROC, SPOOL, CIPARM, GROUP, SYSOUT などである. これら情報は, 通常幾通りか準備して (たとえば, 日中シスト用とか夜間シフト用), オペレーティング・システムのパラメータ・ライブラリに格納しておく.

システムの始動時に, オペレーティング・システムが初期化されたあとで, JES 3 の初期化が行われるが, そのさいにここで規定したパラメータが用いられ運行システムが定義される. この初期化のことを**冷いスタート** (cold start) という. この初期化の状態は, チェックポイントとして DASD に記憶される. したがって, 以後の再始動では, 格別の変更がないかぎり, JES 3 の初期化の過程を省略することができる. かつ, この方法が通常でもある. このようなチェックポイントをもとにした初期化を**暖いスタート** (warm start) という. この場合, スプール上にあるジョブ待行列が, 前に保存してあったままの形で使用されるのである.

2) パラメータの変更

JES 3 コンソールは, ジョブ処理上の種々のコントロールを行うほかに, 初期化パラメータ (MAINPROL, SPOOL, CIPARM など) を実行時に動的に変更することを可能にする. この目的のために, 指令系の中に**照会** (inquiry) および**変更** (modify) の 2 指令が設計されている. 初期化段階で指定したパラメータは, まず照合指令で確認し, 変更指令で変更することができる. これらの指令で処理可能な項目は, つぎのようなものである:

- ローカルや入出力装置を切り離したり, 接続したりする
- CI/MAIN/OUTSERV の各パラメータを変更する
- コンソールの接続関係を変更する

3) 制御ステートメント

JES 3 初期化パラメータは, JES 3 の走行中に暗示的に適用される場合と, ユーザが明示的に指示することによって, はじめて利用可能になる場合の 2 つがある. 後者の例としては, **依存ジョブ処理** (dependent job processing: あるジョブ処理が終了しないと, 別の関係ジョブの処理を開始させないといった処

理方法．これは従来のシステムではオペレータが制御したが，JES 3 では指示にひとづいて自動的に処理してくれる）がある．すなわち，JES 3 は，ユーザがその利用を指示することによって，はじめてその機能が活化する．この種の指示のために，JES 3 制御ステートメントが用意されている．これらのステートメントは，//JOB カードと最初の //EXEC カードの間に挿入する．JES 3 のジョブの流れを読み込む段階で，JES 3 制御ステートメントが解析され，当該ジョブに関する初期化パラメータの一時的変更，あるいは潜在機能の一時的活化のために，以後のステップで使用される．

制御ステートメントによる指示内容には，下記のようなものが掲げられる：

- 特定のローカル・システムを選択すること
- 依存ジョブ処理の指定
- 締切ジョブ処理の指定
- 装置セット・アップの変更
- SYSOUT の種々の仕様の変更

四．回復および再始動機能

OS および JES 3 の双方に，回復および再始動機能が用意されなければならないが，JES 3 が加わったことによって，OS のこの種の機能は何ら影響されず，従前と同様で変更はない．

JES 3 については，そのサービスが複合システム全体に影響することに鑑み，特別な方法が講じられる必要がある．以下でそれについてふれる．

1. DSP のフェール・ソフト

DSP は，ジョブの各段階の処理を担当しているがゆえに，それが異常終止（ABEND）してしまうことについては，深く留意しなければならない．つまり，DSP のプログラム・チェックなどは，そのまま異常終止させてしまうと，関係ないジョブまでが，異常終止にまき込まれることになる．このために，初

期化の最終段階で JES 3 を活化する前に，すべての DSP に対して，異常事態に遭遇したときの共通のエグジットを設定する．この結果，DSP で生じたプログラム・チェックなどは，異常終止モニタの監視をうけ，回復のための新しい DSP のコピーのロードの可否や他の DSP に及ぼす影響などが検査される．異常終止モニタによって救済できないジョブでは，初期化パラメータないしは制御ステートメントの指示にしたがって，強制停止される．なお，指示の種類としては，直ちに最初より再試行，保留（hold）あるいは SYSOUT を印刷してからキャンセルなどが用意されている．これらの処理は，当該ジョブのみに関係し，他のジョブには無関係である．

2. 再始動用スタート機能

再始動用スタートとしては，**熱いスタート**（hot start），**暖かいスタート**（warm start），**冷いスタート**（cold start）の 3 種類がある．

熱いスタートは，グローバルが停止したときのために用意してある．このスタートは，暖いスタートや冷いスタートとは異なり，初期化を行わず，かつローカルに対して何らの影響も与えず，チェックポイントをもとにグローバルを回復しようとするものである．

熱いスタートの試みによって複合システムの回復が不可能な場合には，暖いスタートが試行される．このスタートを行ったときは，スタート時のすべてのジョブは失われてしまうが，スプール上のジョブ待行列だけは保存される．ジョブ待行列の内容が変更されるつど，ジョブ待行列がチェックポイント・レコードに記録されているからである．このスタートには，特定のパラメータをコンソールから与えて，同時にエラーの分析をさせることも可能である．

冷いスタートの典型的な使用法は，すでに前節でふれたが，この方法はエラー回復のために利用することもできる．この場合，スプール・レコードは初期化パラメータにしたがって**初期化**（formatting）されるので，ジョブ待行列の内容は失われてしまう．

3. 動的システム変換

複合システムの中でグローバルの役目は重大である．たとえ，ローカルが健在であっても，グローバルがダウンしてしまうと，途端に複合システムは走行できなくなってしまう．このために，グローバルのダウンに対しては熱いスタートの他に，**動的システム変換**（Dynamic System Interchange；略して **DSI**）の機能が用意されている．

熱いスタートが不成功に終った場合，オペレータはあらかじめ定めておいた交替用プロセッサ（ローカル）の OS 用コンソールから，DSI 指令を出す．この時点で，旧グローバルに対するローカルからの要請はすべてペンディングされ，待機のモードになる．新しく定義されたグローバルは，旧グローバルの状況記録を読み取り，新グローバルの機能を果すようになる．

この DSI を可能にするためには，想定されるローカルにあらかじめスプール，チャネル対チャネル付加装置（CTC），JES3 コンソールなどの接続を考慮しておかなければならない．つまり，物理的接続と初期化パラメータの両方を用意しておく必要がある．

DSI のオペレーションは，OS 用コンソールから DSI の始動を要請したあとは，すべての指示が JES 3 によってコンソール上に打ち出される仕組みになっているので，その誘導にしたがっておこなえばよい．

五．その他の機能

これまで JES 3 の主要な機能についてふれてきたが，ここではこれまでにふれることのできなかった JES 3 の付加的な機能について簡単にまとめておく：

1.　ユーティリィティ・プログラム機能

カードから磁気テープへのコピー，磁気テープからの印刷などといった周辺オペレーションは，DSP によって実現することができる．前述のジョブ処理用 DSP の機能を参照すれば，この考え方は極めて自然である．この種の DSP 機能の利点は，通常のユーティリィティ・プログラムと違って，システム・オーバヘッドが少ないこと，実行優先順位が高いこと（JES 3 の優先順位），コン

ソールから直接始動・制御などができることにある．

2. デバックのための援助機能

JES 3 個有のコントロール制御ブロックの追跡と印刷，特定データ域の印刷機能などが用意されている．また，初期化パラメータは，かなり数が多いので，そのための診断機能も準備されている．

3. 旧システムからの移行機能

移行を容易にするために，JES 3 の前身である **ASP** (Attached Support Processor) をローカルとしてサポートしている（ただし，この場合はローカルと呼ばず，**メイン** (main) と呼んでいるが)．したがって，ASP のユーザはグローバルを新たに設定するだけで，簡単に JES 3 へ移行することができる．

もっと大がかりな移行は，ASPAID という機能を使って行われる．この場合は，JES3 のグローバルと ASP サポート（グローバルに相当）とを，チャネル対チャネル付加装置 (CTC) で接続することによって，ASP ジョブの一部または全部を JES 3 システムに送り込み，実行させることができる．

4. その他の諸機能

前述の 3 つの機能以外に，つぎのような機能も用意されている：

① **TSO** (Time Sharing Option)/**RJE** (Remote Job Entry)：これら機能は，従来の OS と全く同様に JES 3 でもサポートされている．

② **ネットワーク・ジョブ処理**：遠隔地に設定されている JES 3 同志を通信回線で結ぶことにより，ジョブの全部あるいは一部のジョブ・セグメントを別の JES 3 システムに送信し，実行させることができる．これは，主として負荷の均一化のために用いられる．

③ **ユーザ作成 DSP およびエグジット**：すでにふれたように，ジョブ処理に必要な DSP は，JES 3 の基本的約束に従うかぎり，ユーザが変更・追加することができる．また，ジョブ処理の各段階に JES 3 のエグジットがあらかじめ用意されており，これを利用することによって，標準的処理に対する変更，追加あるいは削除を行うことができる．いずれも，ユーザが標準仕様以外のものを必要としたとき，JES 3 をインラインに変更する

ことなく，ユーザ向けの特別な DSP に変えることが可能であり，この結果，システムのバージョンが更新されても，無力な労働をかける必要はなくなっている．

これまで，“ゆるやかに結合された”複合システムに関して，設計の立場ではなく利用者の立場を念頭に，その構成と処理機能を中心に言及してきた．しかも，オペレーションの立場からの追求を主体とした．システム設計者やプログラマにとってみれば，複合システムに発展したからといって，何ら特別な配慮は要求されないからだ．オペレータにとってみれば，複数台のコンピュータ・システムに対してあたかもただ一台のハードウェア・システムであるかのごとき対拠ができるので，これは一大変革である．擬似単一システムは，オペレーション体制に大きな変化を促すものである．いずれにしろ，コンピュータ利用が拡大し，複雑化してくると，この種の複合システムは避けられない．緊密に結合された複合システムも普及するであろうが，ゆるやかに結合された複合システムも一般化するであろう．ジッブ管理の最先端にあるともいうべき JES 3 の概念は，今後のオペレーティング・システムにおけるジョブ・スケジューラの１つの姿を示しているものといえる．今後のコンピュータ・システムの構成とソフトウェア（特にオペレーティング・システムの中のジョブ管理機能）の方向を示唆するものとして，複合システムの構成と機能とを簡潔にまとめたのが本章である．

≪著者略歴≫
江村潤朗
えむらじゅんろう

昭和13年　広島県神石郡三和町にて生まれる
昭和37年　早稲田大学政治経済学部卒業
同年4月　日本アイ・ビー・エム株式会社に入社．カストマー教育のインストラクターを約5年間担当．昭和43年よりコンピュータ教育の手法や教材の開発にあたる．
現　在　**日本アイ・ビー・エム(株)**
教育センター・エデュケーション・アドバイザー
著　書　「すぐに役立つコンピュータ入門」（日本工業新聞社）
主要論文　「企業内におけるコンピュータ要員教育——その問題点と将来の方向——」（**情報処理** Vol. 12, No. 11 (1971) p. 731〜734).「コンピュータ要員の育成技法」（ビジネス・コミュニケーション Vol. 11, No. 7〜Vol. 12, No. 10). その他教育工学に関するもの，企業内コンピュータ教育の進め方に関するもの，プログラマの適性に関するものおよびコンピュータのハードウェアとソフトウェアに関するものなどがある．
現住所　川崎市高津区向ケ丘 430
宮前平グリーンハイツ35—104

OPERATING SYSTEMS Vol. Ⅲ

JUNRO EMURA
International Business Machines Corporation of Japan

Published by Nippon Computer Kyokai
Hongo 2-chome 15-13, Bunkyoku, Tokyo 113

オペレーティング・システムへの構造的アプローチ（下巻）

昭和52年1月15日　初版発行　　© 1977　江村潤朗

検　印 廃　止	著者　江　村　潤　朗 発行者　村　越　勝　弘 発行所　**日本コンピュータ協会** 東京都文京区本郷2丁目15 電話　東京 (03)　813—6929 振替口座　東京96313番

乱丁本・落丁本はお買上げ書店にてお取りかえいたします．　　printed in Japan